

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



**AUTENTICAÇÃO DO REMETENTE EM SERVIDOR DE
E-MAIL COM ASSINATURA BASEADA NA IDENTIDADE**

PEDRO JUNIOR ASHIDANI

Uberlândia - Minas Gerais

2009

PEDRO JUNIOR ASHIDANI

**AUTENTICAÇÃO DO REMETENTE EM SERVIDOR DE E-MAIL
COM ASSINATURA BASEADA NA IDENTIDADE**

Dissertação apresentada ao Programa de Pós-graduação em Ciências da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de mestre em Ciências da Computação.

Orientador: Prof. Dr. Jamil Salem Barbar

Uberlândia

2009

Dados Internacionais de Catalogação na Publicação (CIP)

- A825a Ashidani, Pedro Junior, 1962-
Autenticação do remetente em servidor de e-mail com assinatura baseada na identidade / Pedro Junior Ashidani. - 2009.
104 f. : il.
- Orientador: Jamil Salem Barbar.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.
1. Computadores - Medidas de segurança - Teses. 2. Correio eletrônico - Teses. 3. Assinaturas digitais - Teses. I. Barbar, Jamil Salem. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3-78

PEDRO JUNIOR ASHIDANI

**AUTENTICAÇÃO DO REMETENTE EM SERVIDOR DE E-MAIL
COM ASSINATURA BASEADA NA IDENTIDADE**

Dissertação apresentada ao Programa de Pós-graduação em Ciências da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de mestre em Ciências da Computação.
Área de concentração: Redes de Computadores.

Uberlândia, 4 de fevereiro de 2009

BANCA EXAMINADORA

Prof. Dr. Jamil Salem Barbar
Universidade Federal de Uberlândia

Prof. Dr. Luís Fernando Faina
Universidade Federal de Uberlândia

Prof. Dr. João Cândido Lima Dovicchi
Universidade Federal de Santa Catarina

Uberlândia

2009

DEDICATÓRIA

Aos meus pais pelo exemplo de caráter, à Valma pelo incentivo e apoio incondicional ao mestrado e aos meus filhos pela paciência e compreensão nos momentos de ausência.

AGRADECIMENTOS

Agradeço ao Prof. Dr. Jamil Salem Barbar pela amizade, oportunidade e confiança oferecidas despojadamente.

Aos amigos Homero, Liliane, Silvia, Hildor e Lúcia pelas orientações e incentivos.

Aos professores e funcionários do PPG da FACOM.

Aos colegas Ítalo, Fernanda e Márcio pelo apoio e companheirismo durante o mestrado.

Ao Centro Universitário do Planalto de Araxá pelo apoio financeiro concedido.

EPÍGRAFE

“A arte da guerra nos ensina a confiar não na probabilidade de o inimigo não atacar, mas em nossa prontidão para recebê-lo; não na possibilidade de não atacar, mas sim no fato de que tornamos nossa posição inatacável.”

— *A Arte da Guerra*, Sun Tzu

RESUMO

Frente ao grande volume de mensagens não solicitadas que invadem as caixas postais de usuários na *Internet*, pesquisam-se mecanismos que impeçam ou dificultem a entrega destas mensagens.

A autenticação de remetentes, apesar de não evitar diretamente a entrega, estabelece confiança nos endereços permitindo que se usem listas de reputação. Este trabalho propõe o uso de assinatura baseada em identidade como mecanismo de autenticação de remetente.

A autenticação feita no âmbito do servidor, com o uso de técnicas de criptografia, evita desperdício de recursos. A assinatura digital baseada em identidade, com o uso de emparelhamentos sobre curvas elípticas, confere desempenho superior em relação aos mecanismos convencionais.

Palavras-chave: Segurança; Autenticação de Remetente; Assinatura baseada em identidade; *e-mail*; *spam*.

ABSTRACT

Due to the great number of undesired messages which overload our email boxes, mechanisms that prevent or make the delivery of such messages more difficult are being researched.

Although the authentication of senders does not directly prevent the delivery of such messages, it establishes trust in the addresses which allow reputation lists to be used.

This essay proposes the use of identity-based signature as a mechanism of the sender's authentication. The authentication made on the server, with the use of cryptographic techniques, avoids waste of resources. Identity-based digital signature with the use of pairing over elliptic curves is more efficient than the ordinary mechanisms.

Key words: safety, sender's authentication, identity-based signature , email, spam.

COPYRIGHT

Copyright © 2009 Pedro Junior Ashidani.

“The copyright of this thesis rests with the author. No quotations from it should be published without the author’s prior written consent and information derived from it should be acknowledged”.

SUMÁRIO

Lista de Figuras	xii
Lista de Tabelas	xvi
Lista de Acrônimos	.xviii
Lista de Símbolos	xx
1 Introdução	1
2 Segurança em <i>E-mail</i>	9
2.1 Fundamentos Matemáticos	11
2.1.1 Notação	11
2.1.2 Grupos, Anéis e Corpos	12
2.1.3 Números Primos	13
2.1.4 Teorema de Fermat	14
2.1.5 Teorema de Euler	14
2.1.6 Logaritmos Discretos Módulo n	14
2.1.7 Curvas Elípticas	15
2.1.8 Soma de Pontos em Curva Elíptica	16

2.1.8.1	Curvas Elípticas Sobre $GF(p)$	17
2.1.9	Mapeamento Bilinear e Emparelhamento	18
2.1.10	Emparelhamento de Tate	18
2.2	Criptografia de Chave Pública	19
2.2.1	Sistema RSA	20
2.2.1.1	A Segurança do RSA	21
2.2.2	Criptografia de Curva Elíptica	23
2.2.2.1	Segurança da Criptografia de Curvas Elípticas	24
2.2.3	Funções de <i>Hash</i>	25
2.2.3.1	Algoritmo de <i>Hash</i> Seguro - SHA	27
2.3	Assinatura Baseada em Identidade	27
2.3.1	Esquema de Assinatura Baseada em Identidade de Paterson	29
2.3.2	Esquema de Assinatura Baseada em Identidade de Hess	31
2.4	Autenticação de Remetente	32
3	Correio Eletrônico.....	35
3.1	Protocolo SMTP	35
3.1.1	Iniciação de Seção SMTP	36
3.1.2	Transações de Mensagens	37
3.1.3	Formato da Mensagem	39
3.2	<i>Sendmail</i>	41
3.2.1	As Funções do <i>Sendmail</i>	41
3.2.2	Configuração do <i>Sendmail</i>	44
3.3	Filtrando Mensagens com o <i>MILTER</i>	47
3.3.1	Configurando o <i>MILTER</i> no <i>Sendmail</i>	47
3.3.2	Políticas de Filtragem	48
3.3.3	Comunicação do Filtro com o MTA	48

3.4	Autenticação de Remetentes	50
4	Aspectos de Implementação e Resultados Obtidos.....	55
4.1	Operação do Esquema Proposto	57
4.2	Ambiente de Desenvolvimento	61
4.3	Aspectos de Implantação de Assinatura Digital Baseada em Identidade	62
4.3.1	Configuração de Parâmetros Gerais	63
4.3.2	Servidor de Chaves	66
4.3.3	Assinatura.....	67
4.3.4	Verificação de Assinatura	69
4.4	Testes e Resultados	73
4.4.1	Testes de Desempenho	73
4.4.2	Testes de Segurança.....	76
5	Conclusão.....	77
	Referências.....	80
	Apêndice A – Arquivos de Configurações, <i>Scripts</i> e Programas.....	86
A.1	Sendmail-milter.mc	86
A.2	Arquivo saf_milter.c	87
A.3	Arquivo saf_setup.c	98
A.4	Arquivo assina.c	100
A.5	Arquivo envia-mail.sh.....	103

LISTA DE FIGURAS

1.1	Visão do Sistema de Correio da <i>Internet</i>	2
1.2	Partes de uma Mensagem Eletrônica	3
1.3	Modelo Simples de Criptografia Simétrica	6
1.4	Modelo Simples de Criptografia de Chave Pública	7
2.1	Os Dois Grupos de Ataques Passivos	9
2.2	Os Quatro Grupos de Ataques Ativos	10
2.3	Curva Elíptica para $a = -1$ e $b = 0$	15
2.4	Curva Elíptica para $a = 1$ e $b = 1$	16
2.5	Algoritmo RSA	20
2.6	Esquema de Assinatura Digital	28

2.7	Algoritmo IBS de Paterson	30
2.8	Algoritmo IBS de Hess	32
2.9	Abstração do Processo de Autenticação de Remetente	33
3.1	Esquema Básico do Projeto do SMTP	36
3.2	Iniciação de seção SMTP	37
3.3	Transação de Mensagens do Protocolo SMTP	38
3.4	Conteúdo de Mensagem com Imagem Anexada	40
3.5	Funcionalidades Definidas no Sendmail.cf	42
3.6	Exemplo de um Arquivo <i>Aliases</i>	43
3.7	Lista de Sistema Operacionais Suportados pelo <i>m4</i>	45
3.8	Lista de Agentes de Entrega	46
3.9	Lista de Funcionalidades Disponíveis	46
3.10	Um Conjunto de MTA Interagindo com um Conjunto de <i>MILTERs</i>	49
3.11	Requisições Simultâneas de Dois MTAs	49
3.12	Algoritmo de Operação do MILTER	50

3.13 Funcionamento da Assinatura Digital do OpenPGP	51
3.14 Esquema de Funcionamento do SPF	52
3.15 Esquema de Funcionamento do DKIM	53
4.1 Esquema de Funcionamento de Filtro de Remetente	57
4.2 Esboço de Operação	58
4.3 MUA Requisita ao CA Chave Secreta para Determinada Chave Pública	58
4.4 MTA Requisita ao Filtro a Análise da Mensagem	59
4.5 Processamento da Mensagem	59
4.6 Esquema de Autenticação de Remetente	61
4.7 Algoritmo de Geração de Parâmetros Gerais do Domínio	63
4.8 Parâmetros Públicos P e P_{PUB} em Formato Legível	64
4.9 Alterações no Arquivo localhost.zone	66
4.10 MUA Requisita Chave Secreta do Servidor de Chaves	67
4.11 Algoritmo de Geração de Chaves Secretas	67
4.12 Algoritmo de Assinatura da Mensagem	68

4.13 Código para Assinatura IBS-Paterson	69
4.14 Código para Assinatura IBS-Hess	69
4.15 Código para Verificação de Assinatura IBS-Hess	71
4.16 Código para Verificação de Assinatura IBS-Paterson	71
4.17 Algoritmo de Verificação de Remetente	72
4.18 Código para Verificação de Assinatura IBS-Paterson	73

LISTA DE TABELAS

2.1	Evolução da Fatoração	23
2.2	Comparação do Tamanho de Chaves em Termos de Esforço Computacional	24
2.3	Avanços na Resolução do Problema do Logaritmo Elíptico	25
2.4	Valor de <i>hash</i> SHA-1 de uma Cadeia de Caracteres	25
2.5	Chave Pública RSA	28
2.6	Chaves Públicas RSAXIBS	29
3.1	Itens <i>m4</i> Necessários e Recomendados	45
3.2	Comportamento do <i>Sendmail</i> na Ausência do <i>MILTER</i>	47
4.1	Comparativo no Tamanho de Armazenamento	65
4.2	Campos do Cabeçalho da Mensagem Assinada	68

4.3	Comparativo entre os Esquemas IBS	70
4.4	Tempos de Assinatura e Verificação Core 2 Duo 2GHz	74
4.5	Tempos de Assinatura e Verificação Pentium 4 3.2GHz	74
4.6	Tempos de Assinar e Verificar	75
4.7	Tempos de Geração de Chaves Secretas	75

LISTA DE ACRÔNIMOS

AES	<i>Advanced Encryption Standard</i>
API	<i>Application Programming Interface</i>
ASCII	<i>American Standard Code for Information Interchange</i>
BIND	<i>Berkeley Internet Name Domain</i>
BSD	<i>Berkeley Software Distribution</i>
CA	<i>Certificate Authority</i>
CCA	<i>Chosen Ciphertext Attack</i>
DES	<i>Data Encryption Standard</i>
DKIM	<i>Domain Keys Internet Mail</i>
DNS	<i>Domain Name System</i>
DSA	<i>Digital Signature Algorithm</i>
ECC	<i>Elliptic Curve Cryptography</i>
FTP	<i>File Transfer Protocol</i>
GCC	<i>Gnu Compiler Collection</i>
GMP	<i>GNU Multiple Precision Arithmetic Library</i>
HTML	<i>HyperText Markup Language</i>
IBS	<i>Identity-Based Signature</i>
IETF	<i>Internet Engineering Task Force</i>
IP	<i>Internet Protocol</i>
LES	<i>Lightweight Email Signatures</i>
MD5	<i>Message Digest 5</i>
MILTER	<i>Mail Filter</i>
MIME	<i>Multipurpose Internet Mail Extensions</i>
MIT	<i>Massachusetts Institute of Technology</i>

MTA	<i>Mail Transfer Agent</i>
MUA	<i>Mail User Agent</i>
MUD	<i>Multi User Dungeons</i>
NCP	<i>Network Control Protocol</i>
NIST	<i>National Institute of Standards and Technology</i>
PBC	<i>Pairing Based Cryptography</i>
PGP	<i>Pretty Good Privacy</i>
PKG	<i>Private Key Generator</i>
RFC	<i>Request for Comments</i>
RSA	<i>Rivest, Shamir e Adleman</i>
SCA	<i>Side Channel Attacks</i>
SHA	<i>Secure Hash Algorithm</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SPAM	<i>Spiced Ham - Unsolicited Bulk Email</i>
SPF	<i>Sender Policy Framework</i>
TCP	<i>Transmission Control Protocol</i>
URLs	<i>Uniform Resource Locator</i>

LISTA DE SÍMBOLOS

K	Chave Secreta
K_U	Chave Pública
K_R	Chave Privada
$\mathbb{Z}_q$	Conjunto de inteiros módulo q
$\mathbb{Z}[x]$	Conjunto dos polinômios na variável x , com coeficientes em $\mathbb{Z}$
$\equiv$	Operador de Congruência
$\mathbb{G}$	Grupo
$\mathbb{R}$	Anel
$\mathbb{F}$	Corpo
$\phi(n)$	Função Totiente de Euler
$\hat{e}$	Emparelhamento
$E(K_U, M)$	Texto cifrado da mensagem M com a chave pública K_U
$H(M)$	Função Hash de M

Capítulo 1

Introdução

Com a evolução das redes de computadores nas últimas quatro décadas, foram desenvolvidas tecnologias de transferência de dados cada vez mais sofisticadas e eficientes que permitiram às empresas e às pessoas a troca de informações com maior rapidez e precisão.

Gradativamente, os usuários de computadores perceberam a utilidade da rede e ampliaram o uso pessoal e profissional. A *Internet* passa assim a ocupar, um lugar de destaque na sociedade, mudando definitivamente o cotidiano contemporâneo.

Uma nova abordagem para apresentação dos conteúdos foi adotada com o uso de hipertextos, imagens, sons e poderosos recursos de multimídia e novos conteúdos foram disponibilizados na *Internet* através de localizadores URLs (*Uniform Resource Locator*).

Em razão das diversas redes se revelarem um mercado extremamente promissor, as aplicações de redes foram desenvolvidas, muitas delas extremamente engenhosas e criativas, como e-mail (correio eletrônico), acesso a computadores remotos, transferência de arquivos, resolução de nomes, transferência de hipertexto, mensagem instantânea, compartilhamento de arquivos, telefonia por *Internet* e videoconferência.

Tais aplicações comunicam-se por meio da troca de mensagens, oferecendo e usando diversos serviços sob o paradigma cliente-servidor. Estas comunicações por sua vez, realizam-se por meio de protocolos que são basicamente conjuntos de regras que padronizam o modo de comunicação entre as partes envolvidas [1].

O sucesso do e-mail, desde os primórdios da *Internet* motivou o seu incremento, tornando-o cada vez mais poderoso e funcional. Ainda hoje, é uma das aplicações mais importantes e mais usadas na *Internet*.

Assim como no correio convencional, o e-mail não obriga as partes envolvidas a estar conectadas, sendo que as mensagens são enviadas e recebidas de acordo com a agenda e necessidade dos usuários. É notória a vantagem do e-mail sobre o correio convencional quanto a rapidez, facilidade de uso e custo.

Com poderosos recursos, como formatação HTML (*HyperText Markup Language*), possibilidade de envio de anexos com fotos, documentos eletrônicos, arquivos multimídia e programas executáveis, o correio eletrônico começou a ser usado como ferramenta de mala direta, sendo possível enviar mensagens desejadas e indesejadas, a milhares de destinatários.

As mensagens são trocadas entre os servidores por meio do protocolo SMTP (*Simple Mail Transfer Protocol*). Caso não consiga ser enviada de um servidor para outro, a mensagem permanece em uma fila de mensagens do servidor de origem que tentará transferi-la mais tarde. Geralmente estas novas tentativas são feitas a cada quinze minutos e, se após alguns dias não for transferida com sucesso, a mensagem será descartada e o servidor notificará o remetente por meio de uma outra mensagem de correio.

O correio eletrônico pode ser esquematizado com três principais componentes: clientes de correio, servidores de correio e protocolo SMTP. A Figura 1.1 representa o sistema de correio da *Internet*. Esquematicamente, tem-se um remetente, Alice, que envia uma mensagem eletrônica para um destinatário, Bob. Clientes de e-mail permitem que os usuários leiam, salvem, escrevam, respondam e encaminhem mensagens. Ao terminar de compor sua mensagem, Alice a envia para o seu servidor de correio, que por sua vez, encaminha para o servidor de correio de Bob utilizando o protocolo SMTP. Quando achar conveniente Bob conecta-se ao seu servidor de correio, transfere as mensagens nele armazenadas.

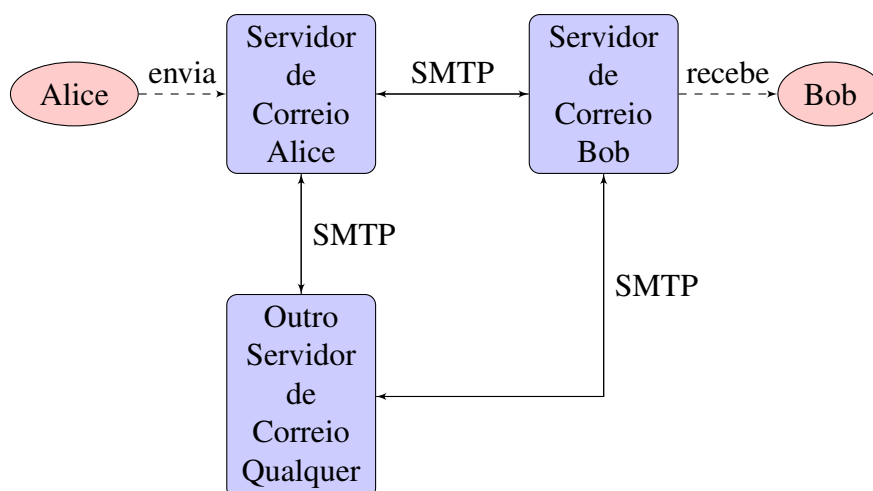


Figura 1.1: Visão do Sistema de Correio da *Internet*

O protocolo SMTP transmite um objeto de correio que possui duas partes: um envelope e

um conteúdo, sendo que o conteúdo é subdividido em duas partes: cabeçalho e dados, conforme pode ser observado na Figura 1.2.

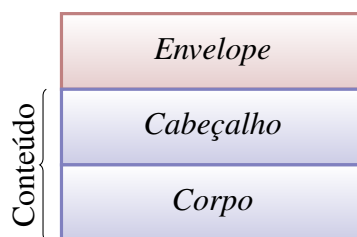


Figura 1.2: Partes de uma Mensagem Eletrônica

As especificações RFC876, RFC2821 e RFC5321¹ [2–4] definem que os dois hospedeiros participantes de uma transação SMTP são denominados cliente SMTP e servidor SMTP. Quando um cliente SMTP inicia uma sessão com um servidor SMTP, trocam mensagens de identificação. Após se identificarem, o cliente envia dois comandos: *MAIL FROM* e *RCPT TO*.

Com o comando *MAIL FROM*, o cliente SMTP envia o endereço eletrônico do remetente, e com o comando *RCPT TO*, o cliente SMTP envia o endereço eletrônico do destinatário. Estas duas informações basicamente compõem o envelope da mensagem. O endereço do remetente recebe algumas verificações de validade, porém sua autenticidade não é verificada. Este fato permite que atacantes troquem seus endereços de remetentes por outros endereços quaisquer válidos, mesmo que estes endereços pertençam a outros usuários clientes *SMTP*.

Devido à simplicidade dos protocolos envolvidos nas transmissões de mensagens eletrônicas, os mecanismos de identificação de *spam* podem ser dificultados ou até mesmo burlados [5]. Apesar de exigir um conhecimento técnico mais avançado, é possível forjar endereços IP (*Internet Protocol*) burlando as listas negras de servidores e as listas negras de redes. Forjar o endereço eletrônico do remetente é uma tarefa bem mais simples, o que torna a lista de remetentes inútil. Em um cenário de tantas mensagens inválidas, a credibilidade do sistema perante o usuário fica comprometida. Torna-se necessário, portanto, a certeza da identidade do remetente [6].

Originalmente, SPAM (*Spiced Ham - Unsolicited Bulk Email*) denomina uma marca de presunto apimentado (*Spiced Ham*) enlatado da *Hormel Foods*, usada repetida e irritantemente por um grupo de comediantes inglês chamado *Monty Python* em um quadro apresentado na TV. Alguns usuários do MUD (*Multi User Dungeons*) começaram a associar o termo *spam* às mensagens não solicitadas, repetidas e irritantes [7].

¹Os padrões usados na *Internet* são desenvolvidos pela IETF (*Internet Engineering Task Force*) e são armazenados em documentos denominados RFC (*Request for Comments*).

Os administradores de rede e pesquisadores procuram mecanismos para evitar os danos causados pelas mensagens não solicitadas, como ameaças à segurança de rede, ameaça à identidade dos usuários e desperdício de recursos de rede. Esses recursos dizem respeito à capacidade de transmissão dos enlaces, espaço de armazenamento e capacidade de processamento dos servidores.

Em clientes, essas mensagens também consomem recursos dos usuários finais, sendo a mais crítica à capacidade de transmissão do enlace. Deve-se considerar ainda que tais mensagens não solicitadas podem conter vírus, cavalos de tróia² e *keyloggers*³, que constituem graves ameaças à segurança da rede, preocupando os seus administradores.

Estas mensagens indesejadas, inofensivas ou perigosas, geralmente chegam aos usuários finais disfarçadas (*fake mail*). Com a popularização da *Internet*, os usuários finais utilizam os recursos da rede somente com conhecimentos elementares do aplicativo e do sistema operacional. Os atacantes, cientes da fragilidade destes usuários, apontam suas armas para esse elo mais fraco da corrente. Os usuários finais são ludibriados com mensagens contendo chamadas de assuntos curiosos, como fotos de alguma celebridade, promessas de vantagens lucrativas, avisos de cancelamentos de perfis de comunidades de relacionamento ou avisos de faturamentos de lojas virtuais.

Devido ao baixo custo de utilização, em relação ao correio convencional, e a facilidade de se coletar endereços eletrônicos pela *Internet*, muitos atacantes passaram a usar o serviço de mensagens eletrônicas de maneira indevida. As ameaças podem ser simples propagandas comerciais, pirâmides, até cavalos de tróia etc. Programas especializados em coletar endereços pela *Internet* foram criados e agora se têm caixas postais abarrotadas de mensagens indesejadas provenientes de fontes como *spammers*⁴, *fraudsters*⁵ e *virus worm*⁶. Mais de 81% do volume dos e-mails são SPAM [8, 9]. Os ataques crescem em volume [10] e em sofisticação [11, 12]

Inicialmente foram criados filtros baseados no campo *FROM*, usados para proteger as caixas postais dos usuários. Listas Negras, Brancas e Cinza [13] constituem listas de endereços e de hospedeiros usadas para identificar mensagens de remetentes indesejados. Entretanto, atualmente, a maior parte das mensagens não solicitadas possuem o remetente forjado, o que dificulta o uso de listas.

As ameaças contidas nas mensagens eletrônicas muitas vezes provêm de remetente for-

²Programa de computador que uma vez instalado permite a invasão do computador por pessoas não autorizadas.

³Programa de computador que grava teclas usadas

⁴Disseminam mensagens comerciais não solicitadas.

⁵Enviam mensagens com objetivo de realizar fraudes.

⁶Programa nocivo com capacidade de se replicar e espalhar automaticamente.

jado. Não se pode mais confiar na veracidade do endereço do remetente. Caso os endereços forjados pertençam a remetentes legítimos, estes terão sua reputação abalada. Apesar do desenvolvimento de métodos criativos para a garantia da identidade dos remetentes, a criptografia continua sendo o caminho mais adequado a ser seguido pelos pesquisadores.

Desde os tempos da Roma antiga, o homem deseja evitar que suas mensagens possam ser acessadas indevidamente. Confidencialidade, autenticação, integridade, irretratabilidade, controle de acesso e disponibilidade são serviços importantes na área de segurança da informação.

O uso de e-mail demanda principalmente confidencialidade, irretratabilidade e autenticação. A propriedade confidencialidade é imprescindível para que somente os envolvidos na comunicação possam ter acesso ao conteúdo da mensagem. Isto pode ser obtido quando o transmissor cifra a mensagem e o receptor a decifra. A propriedade irretratabilidade é desejável para garantir que o remetente negue a autoria da mensagem enviada ou que o destinatário negue o recebimento de uma mensagem. A propriedade autenticação se faz necessária para que transmissor e receptor possam confirmar a identidade da outra parte envolvida.

Os serviços relacionados à área de segurança fundamentam-se fortemente em técnicas criptográficas. Basicamente, têm-se sistemas criptográficos simétricos e sistemas criptográficos assimétricos. Os sistemas criptográficos simétricos possuem a característica de ser simples de programar e muito rápidos, porém com o empecilho de que as partes devem compartilhar uma chave comum. Já os sistemas criptográficos assimétricos baseiam-se em problemas matemáticos difíceis em que, durante a execução dos algoritmos, operações complexas devem ser realizadas.

Na criptografia simétrica a mensagem transmitida é cifrada através de um algoritmo criptográfico conhecido e público, usando-se uma chave K secreta, e depois decifrada com um algoritmo de decifragem usando-se a mesma chave secreta K , conforme observado na Figura 1.3. Exemplos deste tipo de algoritmo são o DES (*Data Encryption Standard*) [14] e o AES (*Advanced Encryption Standard*) [15]. Apesar do grande sucesso do DES, sua chave de 56 bits tornou-se pequena diante do poder computacional dos dias de hoje. Para elucidar esse poder, pode-se citar a quebra do DES, em 1997 no DES Challenge⁷, que foi alcançada em um esforço coletivo em menos de quatro meses e em 1999, no DES Challenge III, o vencedor conseguiu descobrir a chave em pouco mais de 22 horas [16, 17].

Diante da insegurança do DES surge o AES. Ele pode trabalhar com chaves de 128, 192 e 256 bits e seu algoritmo processa os dados em blocos de 128 bits. Para ilustrar da força do AES, estima-se que uma máquina que pudesse, através de força bruta, quebrar o DES de 56 bits

⁷Desafio proposto pelo *RSA Security* para evidenciar a falta de segurança do DES

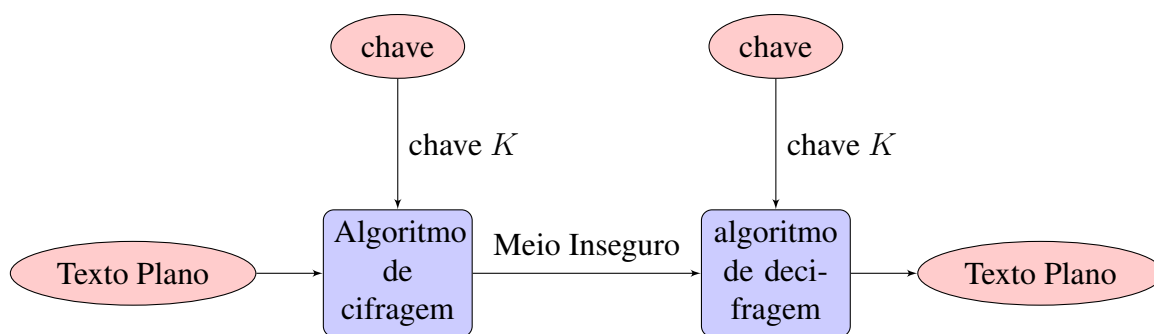


Figura 1.3: Modelo Simples de Criptografia Simétrica

em 1ms, levaria 149 bilhões de anos para quebrar o AES com uma chave de 128 bits.

Somente os SCA (*Side Channel Attacks*) tiveram algum sucesso contra o AES [18]. Para este tipo de ataque, o oponente necessita acesso ou proximidade ao computador do participante. Como exemplo, podem-se citar dados coletados da radiação eletromagnética do processador, tempo de execução do algoritmo de cifragem entre outros, que podem dar pistas e reduzir o espaço de busca da chave criptográfica.

O AES baseia-se em uma rede de substituições apresentando as características de simplicidade, uso de pouca memória e desempenho rápido tanto em *software* como em *hardware*. É atualmente adotado pelo governo dos EUA para proteção das transmissões de dados normais (chave de 128 bits), classificados e altamente classificados (chaves de 192 e 256 bits). Um grande problema é como duas partes podem compartilhar a mesma chave através de um canal inseguro. Para contornar o problema da existência de um canal inseguro, utiliza-se a criptografia assimétrica para troca de chaves.

A criptografia assimétrica, conhecida por criptografia de chaves públicas, como ilustrada na Figura 1.4, é assim denominada porque o receptor dos dados possui uma chave pública (K_U) que qualquer um pode usar para criptografar empregando um algoritmo pré-determinado. Somente o receptor possui a chave privada (K_R) capaz de decifrar a mensagem cifrada pelo transmissor dos dados. Como a chave de cifrar é diferente da chave de decifrar diz-se que é um sistema criptográfico assimétrico. A vantagem deste sistema é que uma das chaves pode ser divulgada livremente sem a necessidade de um canal seguro, diferentemente do sistema simétrico no qual a chave secreta K deve ser compartilhada por ambos os lados participantes da comunicação.

O projeto do protocolo SMTP [3, 4] permite que remetentes informem, no envelope da mensagem eletrônica, qualquer endereço para o campo FROM, o que torna difícil a autenticação do remetente. Algumas soluções como o PGP (*Pretty Good Privacy*), OpenPGP [19] e o

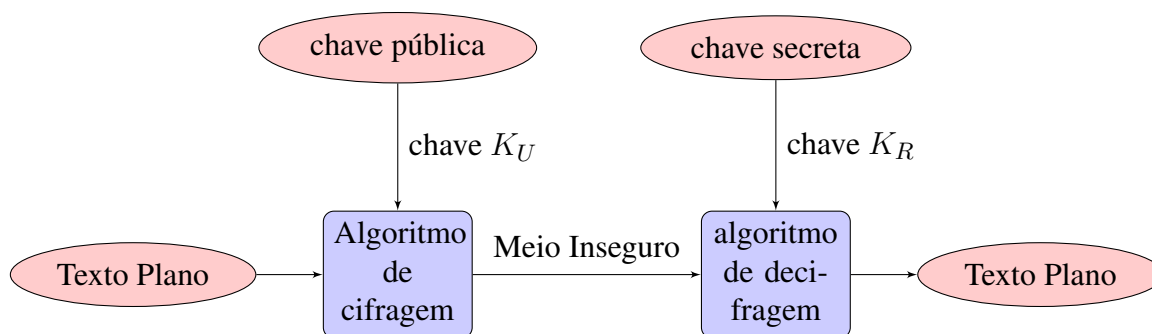


Figura 1.4: Modelo Simples de Criptografia de Chave Pública

LES (*Lightweight Email Signatures*) [20] permitem que o cliente verifique a identidade do remetente; porém, como nestes casos, a mensagem deve chegar até o cliente para a verificação, tem-se o desperdício de recursos de rede, armazenamento e tempo.

Deve-se ressaltar que o alto volume de mensagens indesejadas [8] constitui um problema cujas possíveis soluções merecem ser exploradas e desenvolvidas. Assim, é necessário um mecanismo que permita identificar mensagens forjadas antes que as mensagens sejam entregues aos clientes, evitando o desperdício de recursos, sem onerar demasiadamente o servidor SMTP.

Em face deste contexto, esta dissertação tem como objetivo propor um mecanismo para autenticação do remetente, em que a verificação da identidade seja feita no âmbito do servidor. Para tal, é necessário que o sistema de assinatura da mensagem, feita pelo cliente, tenha uma chave pública fácil de ser determinada e um custo computacional similar ou inferior aos mecanismos tradicionais de assinatura digital. Este mecanismo deve ainda permitir uma fácil implantação nos servidores SMTP, com alteração somente nas configurações dos aplicativos existentes.

Para a realização de testes e desenvolvimento do mecanismo proposto foram utilizados dois computadores com o sistema operacional Linux conectados através de uma rede local. A aplicação do servidor SMTP usado foi o Sendmail e a aplicação cliente utilizada foi o Thunderbird. Para a autenticação do remetente utilizou-se um Milter que é uma extensão do Sendmail para filtrar mensagens indesejadas.

A escolha do servidor como responsável pela autenticação da identidade do remetente justifica-se pela economia de recursos de infra-estrutura (banda de transmissão e armazenamento) obtidos em relação às propostas de verificação da identidade fim-a-fim. O uso da verificação da identidade do remetente no servidor esbarra na dificuldade de se encontrar a chave pública do remetente ou na verificação de listas de certificados revogados. O uso de assinatura baseada em identidade proposta nesta dissertação resolve estas questões.

Com o servidor SMTP como responsável pela autenticação de vários remetentes, a questão do custo computacional de cada verificação torna-se relevante. por essa razão optou-se pelo esquema de IBS (*Identity-Based Signature*) que faz uso de ECC (*Elliptic Curve Cryptography*) o qual possui um custo computacional menor [21] que o RSA (*Rivest, Shamir e Adleman*) [22] e o DSA (*Digital Signature Algorithm*) [23].

A esta introdução, seguem-se quatro capítulos, descritos resumidamente a seguir.

No Capítulo 2, apresenta-se conceitos relacionados com segurança e criptografia, além de conceitos de criptografia de chaves públicas e IBS. Também se apresenta os conceitos matemáticos que fundamentam a criptografia assimétrica.

No Capítulo 3, apresenta-se informações sobre o correio eletrônico, tais como definições, arquitetura e protocolos, aplicativos e seus módulos, utilizados pelos aplicativos de correio eletrônico. Neste capítulo expõem-se ainda mecanismos utilizados para autenticação de remetente.

O Capítulo 4 apresenta a proposta para autenticação de remetente em servidor de e-mail com assinatura baseada em identidade. Discute-se mecanismos de publicação dos parâmetros e de definição da assinatura, algoritmo para assinatura e algoritmo para verificação. Apresenta-se também os resultados obtidos com os testes executados para validação.

O Capítulo 5 ao concluir o trabalho, apresenta sugestões para aprimoramento da pesquisa realizada, aponta seus pontos positivos e negativos, além de apresentar o levantamento da contribuição deste estudo à comunidade científica e acadêmica e propostas para futuros trabalhos.

Capítulo 2

Segurança em *E-mail*

Com o crescimento e popularização da informação digital, são usados novos mecanismos de armazenamento e transmissão. As ameaças à segurança da informação apresentam-se agora em versões digitais. Os ataques classificam-se basicamente em dois grupos: ataques passivos e ataques ativos [24].

Na Figura 2.1a o adversário Trudy intercepta e lê a mensagem enviada por Bob para **Alice**. Já na Figura 2.1b as mensagens enviadas por Bob para Alice estão disfarçadas de maneira que Trudy não consiga entender. Porém, Trudy pode ter pistas do conteúdo analisando a frequência e o tamanho das mensagens.

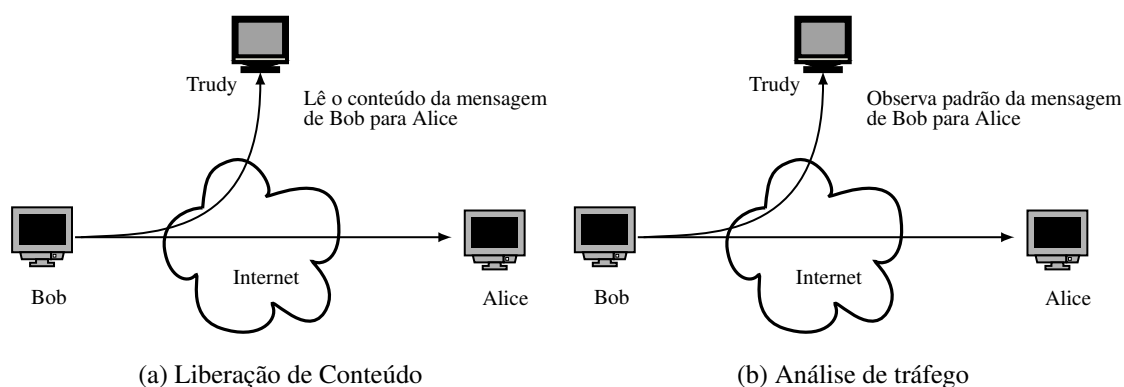


Figura 2.1: Os Dois Grupos de Ataques Passivos

Nos ataques ativos tem-se alguma modificação na ordem das mensagens ou na criação de uma mensagem falsa, dividem-se em quatro grupos: personificação, repetição, modificação e negação, como mostra a Figura 2.2

Uma personificação trata-se da utilização da identidade de uma entidade por outra. Este

ataque geralmente é realizado juntamente com outros tipos de ataques ativos.

A repetição envolve uma captura passiva de uma mensagem que é depois retransmitida produzindo um efeito não autorizado.

Na modificação do fluxo de informação, alguma parte de uma mensagem legítima foi alterada ou a ordem das mensagens foi mudada para produzir um efeito não autorizado.

A negação de serviço impede ou dificulta o uso de serviços de rede. Este ataque pode ter um alvo específico como um servidor *Web* ou pode ter como alvo uma rede inteira.

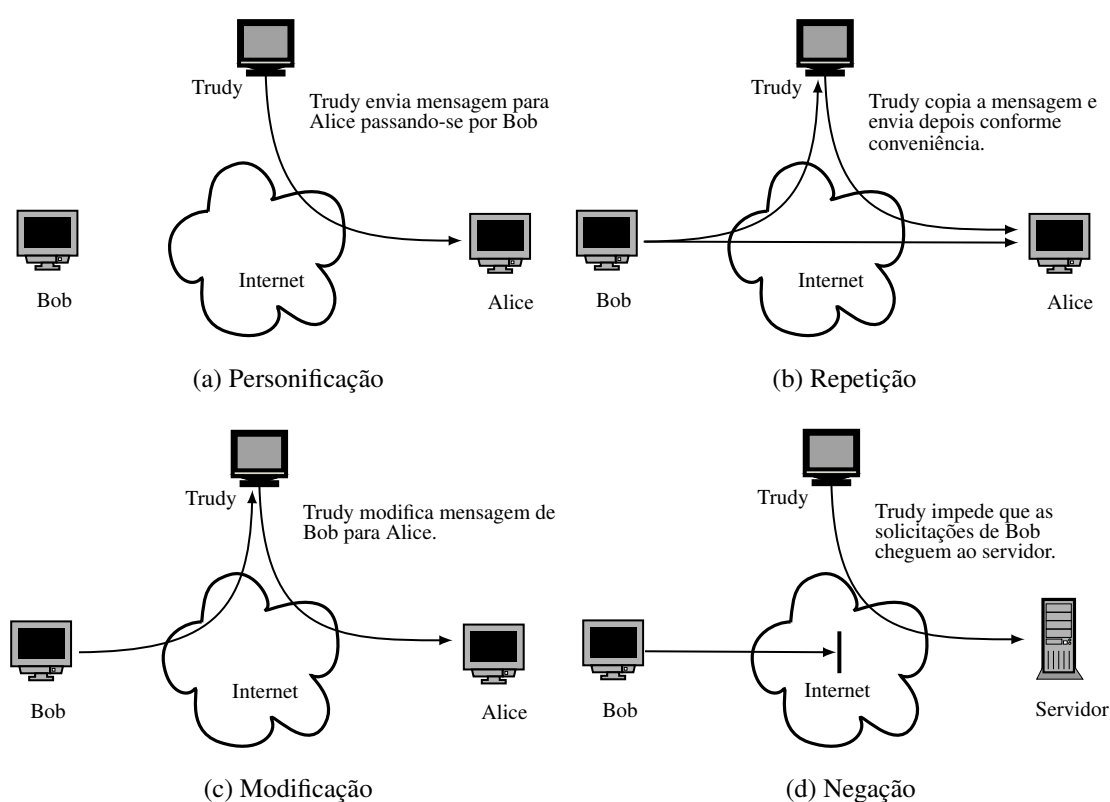


Figura 2.2: Os Quatro Grupos de Ataques Ativos

Os ataques passivos possuem características diferentes dos ativos. Os ataques passivos são difíceis de ser detectados. Portanto, alguns serviços de segurança devem ser usados para impedir o seu sucesso. Os ataques ativos, em contrapartida são difíceis de ser evitados. Dessa forma, o objetivo das medidas de segurança torna-se detectar o ataque para resguardar-se de atrasos ou interrupções ou, ainda recupera-los. Um efeito profilático pode ser obtido caso o mecanismo de detecção seja intimidador.

Com o intuito de prover proteção de algum tipo específico aos recursos do sistema utilizam-se os serviços de segurança classificados como: confidencialidade, autenticação, controle de

acesso, integridade de dados, irretratabilidade e disponibilidade [18, 25].

O serviço de autenticação está relacionado com garantia de que uma comunicação é autêntica. Pode-se ter a garantia da identidade somente de uma das partes da comunicação, por exemplo, quando se acessa um servidor *web* que deseja garantir aos seus usuários a identidade do servidor, ou pode-se ter a garantia da identidade de ambas as partes, como um usuário usando um terminal remoto, em que tanto o servidor como usuários precisam garantir que possuem a identidade que eles afirmam ter.

A integridade de dados aplica-se a um fluxo de dados, partes de uma mensagem ou uma mensagem inteira. Este serviço tem como objetivo garantir que os dados sejam entregues na ordem correta, sem duplicação e sem modificação do seu conteúdo.

A irretratabilidade impede ao emissor e ao receptor negar que a mensagem foi transmitida. Por meio deste recurso, quando uma transmissão é efetuada, o receptor pode provar que o emissor realmente a realizou. Da mesma forma, o emissor pode provar que o receptor de fato recebeu a transmissão.

Disponibilidade é uma propriedade de um sistema estar utilizável e acessível, de acordo com as especificações de desempenho do serviço que se deseja oferecer.

Para a autenticação do remetente, os serviços de autenticação e integridade são fundamentais. Estes serviços baseiam-se nos conceitos de criptografia de chaves públicas, que utilizam os conceitos matemáticos descritos a seguir.

2.1 Fundamentos Matemáticos

Os fundamentos matemáticos são necessários para o entendimento de como é desenvolvida a segurança na troca de informações. Por meio deles, é possível a compreensão dos conceitos de criptografia de chave pública.

2.1.1 Notação

Seguem-se algumas notações usadas nesta dissertação:

- ✓ $\mathbb{Z}_q$ representa o conjunto de inteiros módulo q , isto é, $\{0, 1, \dots, q-1\}$, onde o resultado das operações aritméticas elementares de $\mathbb{Z}$ é reduzido módulo q ;
- ✓ $\mathbb{Z}_q^* = \mathbb{Z}_q \setminus \{0\}$;

- ✓ $\mathbb{Z}[x]$ denota o conjunto dos polinômios na variável x , com coeficientes em $\mathbb{Z}$;
- ✓ $\mathbb{G}^* = \mathbb{G} \setminus \{O\}$, onde O denota o elemento identidade do grupo $\mathbb{G}$;
- ✓ $\{0, 1\}^n$ denota o conjunto de cadeias binárias de comprimento n ;
- ✓ $\{0, 1\}^*$ denota o conjunto de cadeias binárias de comprimento arbitrário.
- ✓ $a \equiv b \pmod{n}$ denota que a é congruente a b módulo n ou seja, n é divisível por $(a - b)$ e $\equiv$ é o operador de congruência.

2.1.2 Grupos, Anéis e Corpos

Um **grupo** $\mathbb{G}$ é um conjunto de elementos com uma operação binária, indicada por $\circ$, que associa a cada par ordenado (a, b) de elementos em $\mathbb{G}$ um elemento $(a \circ b)$ em $\mathbb{G}$, de modo que os seguintes axiomas, denominados G1, G2, G3 e G4, são obedecidos [25]:

- (G1) Fechamento: $\forall a, b \in \mathbb{G}, a \circ b \in \mathbb{G}$;
- (G2) Associatividade: $\forall a, b, c \in \mathbb{G} : (a \circ b) \circ c = a \circ (b \circ c)$;
- (G3) Elemento identidade: $\exists e \in \mathbb{G} : \forall a \in \mathbb{G} : a \circ e = e \circ a = a$;
- (G4) Elemento inverso: $\forall a \in \mathbb{G} : \exists \bar{a} \in \mathbb{G} : a \circ \bar{a} = e$.

Um grupo com uma quantidade finita de elementos é chamado de **grupo finito** e a quantidade de elementos do grupo é chamada de **ordem** do grupo.

Chama-se **abeliano** o grupo que satisfizer a seguinte condição adicional, denominada G5:

- (G5) Comutativo: $\forall a, b \in \mathbb{G} : a \circ b = b \circ a$.

Chama-se de **cíclico** o grupo formado por elementos resultantes da aplicação repetida do operador do grupo de modo que $a^3 = a \circ a \circ a$. Também se define que $a^0 = e$, o elemento de identidade; e $a^{-n} = (a')^n$. Um grupo $\mathbb{G}$ é cíclico se os elementos de $\mathbb{G}$ forem uma potência a^k (k é um inteiro positivo) de um elemento fixo $a \in \mathbb{G}$. Considera-se que o elemento a é o gerador do grupo $\mathbb{G}$. Um grupo cíclico é sempre abeliano.

Define-se um **anel** $\mathbb{R}$ como um conjunto de elementos com duas operações binárias, chamadas adição e multiplicação, de forma que para todo $a, b, c \in \mathbb{R}$, os seguintes axiomas, denominados A1, A2 e A3, são obedecidos:

(G1-G5): $\mathbb{R}$ é um grupo abeliano com relação à adição, ou seja, $\mathbb{R}$ satisfaz os axiomas de G1 a G5;

(A1) Fechamento sob multiplicação: $\forall a, b \in \mathbb{R} : a \times b \in \mathbb{R}$;

(A2) Associatividade da multiplicação: $\forall a, b, c \in \mathbb{R} : a.(b.c) = (a.b).c$;

(A3) Leis distributivas: $\forall a, b, c \in \mathbb{R} : a \times (b + c) = a \times b + a \times c$
 $\forall a, b, c \in \mathbb{R} : (a + b) \times c = a \times c + b \times c$;

Um anel é considerado **comutativo** se satisfizer uma condição adicional:

(A4) Comutatividade da multiplicação: $\forall a, b \in \mathbb{R} : a \times b = b \times a$.

Define-se como um **domínio de integridade** o anel comutativo que obedece aos seguintes axiomas:

(A5) Identidade multiplicativa: $\exists 1 \in \mathbb{R} : a \times 1 = 1 \times a$

(A6) Sem divisores zero: Se a, b em $\mathbb{R}$ e $ab = 0$, então $a = 0$ ou $b = 0$ ou $a = b = 0$.

Um **corpo** $\mathbb{F}$ é um conjunto de elementos com duas operações binárias, chamadas de *adição* e *multiplicação*, de modo que para todo $a, b, c \in \mathbb{F}$, os seguintes axiomas são obedecidos:

(G1-A6): $\mathbb{F}$ é um domínio de integridade;

(A7) Inverso Multiplicativo: Para cada $a \in \mathbb{F}^*$, $\exists a^{-1} \in \mathbb{F} : a \times a^{-1} = (a^{-1}) \times a = 1$.

2.1.3 Números Primos

Diversos conceitos da Teoria dos Números são importantes para a criptografia de chaves públicas, especialmente o conceito de números primos. Um número primo, por definição, é um número p inteiro e $p > 1$, o qual não possui divisores positivos inteiros diferentes de 1 e dele mesmo [26].

Se $z > 1$ e z não é primo, diz-se que z é composto. Ou seja, diz-se que z é composto se e somente se $z = ab$, onde $\exists a, b \in \mathbb{N}$ com $1 < a < z$ e $1 < b < z$ [27].

O Teorema Fundamental da Aritmética diz que todo inteiro diferente de zero pode ser expresso como

$$z = \pm p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_n^{e_n} \quad (2.1)$$

onde $p_1, \dots, p_n$ são primos distintos e $e_1, \dots, e_n$ são inteiros positivos [27].

Define-se primos relativos ou coprimos como dois números inteiros $a, b \in \mathbb{Z}$ tal que o máximo divisor comum seja igual a ± 1 ou seja, $\text{mdc}(a, b) = 1$.

2.1.4 Teorema de Fermat

Diz o teorema de Fermat [25]: Se p é um número primo e a é um inteiro positivo tal que a e p são primos relativos, então:

$$a^{p-1} \equiv 1 \pmod{p} \quad (2.2)$$

É útil conhecer esta forma alternativa do teorema de Fermat: Se p é primo e a é um inteiro positivo, então [25]:

$$a^p \equiv a \pmod{p} \quad (2.3)$$

2.1.5 Teorema de Euler

Diz o teorema de Euler [25]: Seja a e n primos relativos, então:

$$a^{\phi(n)} \equiv 1 \pmod{n} \quad (2.4)$$

onde $\phi(n)$ é a função totiente de Euler, que retorna a quantidade de números inteiros positivos menores que n e primos relativos de n . Matematicamente temos:

$$\phi(n) = |\{a \in \mathbb{N} \mid a < n \wedge \text{mdc}(n, a) = 1\}| \quad (2.5)$$

2.1.6 Logaritmos Discretos Módulo n

Ao trabalhar-se com números reais, o cálculo de a^x não é significativamente mais fácil [28] que a operação inversa $\log_a x$. Em um grupo finito $\mathbb{F}_q^*$ é possível também calcular rapidamente $a^x \pmod{p}$ para valores grandes de x pelo método da repetição quadrática. Porém, dado um valor y , o qual pode ser representado na forma $a^x \pmod{p}$, é difícil calcular x [29] onde x é chamado de logaritmo discreto de b base a e é representado como $\log_a b$ e:

$$a^x \equiv b \pmod{n} \quad (2.6)$$

2.1.7 Curvas Elípticas

As curvas elípticas possuem este nome porque são descritas por equações cúbicas, parecidas com as usadas por uma elipse, mas não são elipses. As curvas elípticas possuem a seguinte forma geral [25, 26, 30, 31]:

$$y^2 + axy + by = x^3 + cx^2 + dx + e \quad (2.7)$$

onde a, b, c, d e e são números reais. Também se insere na definição da curva elíptica um ponto O chamado *ponto no infinito*. Pode-se simplificar a equação 2.7 para a forma:

$$y^2 = x^3 + ax + b \quad (2.8)$$

Para determinados valores de a e b para cada valor de x , tem-se dois valores de y (um positivo e outro negativo) que satisfaz a equação 2.8. Assim o conjunto de pontos (x, y) formam uma curva simétrica em relação ao eixo x . A Figura 2.3 mostra um exemplo de curva elíptica para $a = -1$ e $b = 0$.

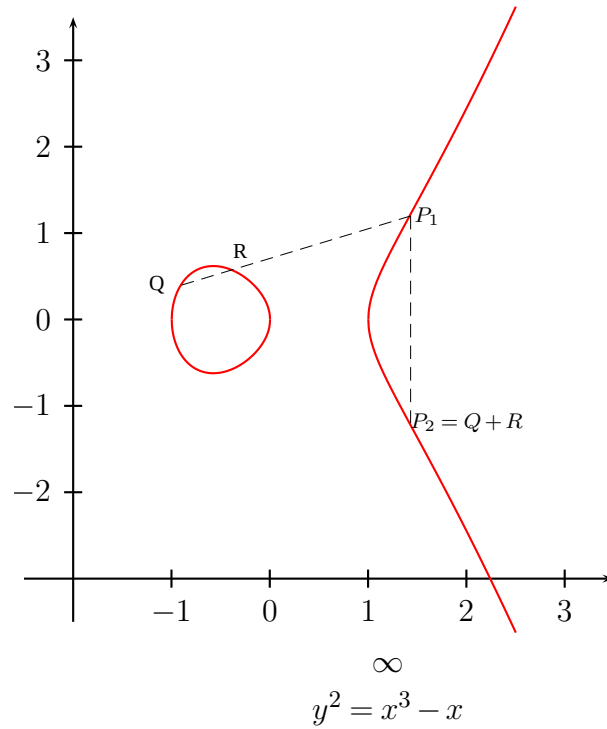


Figura 2.3: Curva Elíptica para $a = -1$ e $b = 0$

Seja $E(a, b)$ o conjunto de todos os pontos (x, y) que satisfazem a equação 2.8 mais o elemento O , então diferentes valores de (a, b) produzem diferentes conjuntos de pontos $E(a, b)$. A Figura 2.3 representa os pontos $E(-1, 0)$, enquanto a curva da Figura 2.4 representa os pontos $E(1, 1)$.

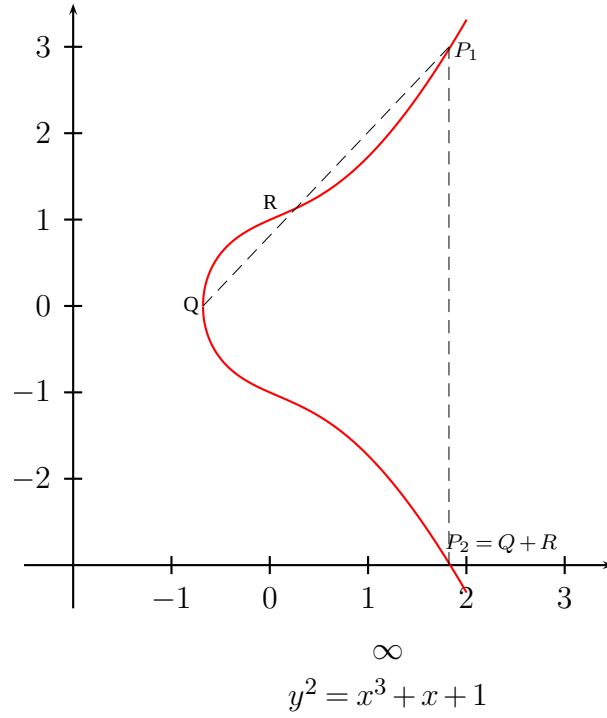


Figura 2.4: Curva Elíptica para $a = 1$ e $b = 1$

2.1.8 Soma de Pontos em Curva Elíptica

Stallings demonstrou que um grupo pode ser definido com base nos pontos $E(a, b)$ para valores específicos de a e b na equação 2.8, desde que a condição 2.9 seja atendida [25]:

$$4a^3 + 27b^2 \neq 0 \quad (2.9)$$

Para definir um grupo, define-se uma operação, chamada de adição e representada por $+$, para o conjunto $E(a, b)$, onde a e b satisfazem a condição 2.9.

A adição pode ser definida resumidamente da seguinte forma:

1. Se três pontos pertencentes à curva estão em linha reta sua soma é igual a O
2. O é o elemento neutro da adição, pois $O = -O$ e $P + O = P$
3. Se uma linha vertical encontra a curva em dois pontos na mesma coordenada x , – por exemplo $P_1 = (x, y)$ e $P_2 = (x, -y)$, a reta encontra no infinito também o ponto O , portanto $P_1 + P_2 + O = O$, assim $P_1 = -P_2$
4. Para somar dois pontos Q e R de diferentes abscissas x , traçar uma reta entre os pontos Q e R e encontrar a intersecção da reta com a curva no ponto P_1 , é fácil notar que há somente um único ponto P_1 . Assim, $P + Q + P_1 = O$ então $P + Q = -P_1$, conforme observado na Figura 2.4
5. Para calcular o dobro do ponto Q , traça-se uma tangente ao ponto Q que faz intersecção com a curva elíptica em S . Assim, $Q + Q = -S$ e $2Q = -S$.

Logo para multiplicar um ponto P por um inteiro k , deve-se somar k vezes o ponto P . Desta modo, $P + P + P + P = 4P$.

Algebricamente, tem-se que para dois pontos distintos $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$ que não são negativos um do outro, a inclinação Δ da linha l que os une é:

$$\Delta = \frac{(y_Q - y_P)}{(x_Q - x_P)} \quad (2.10)$$

Pode-se expressar as coordenadas do ponto $R = P + Q$ da seguinte forma:

$$\begin{cases} x_R = \Delta^2 - x_P - x_Q \\ y_R = -y_P + \Delta(x_P - x_R) \end{cases} \quad (2.11)$$

Para o caso da soma de um ponto a ele mesmo, $P + P = 2P = R$, para $y_P \neq 0$ tem-se:

$$\begin{cases} x_R = \left(\frac{3x_P^2 + a}{2y_P} \right)^2 - 2x_P \\ y_R = \left(\frac{3x_P^2 + a}{2y_P} \right) (x_P - x_R) - y_P \end{cases} \quad (2.12)$$

2.1.8.1 Curvas Elípticas Sobre $GF(p)$

Duas famílias de curvas elípticas são usadas em aplicações criptográficas: curvas sobre $GF(p)$ e curvas sobre $GF(2^m)$. Segundo Fernandes [32] $GF(p)$ é melhor para aplicações de *software*, devido às operações de *bit* não serem necessárias, enquanto que $GF(2^m)$ é me-

lhor para aplicações de *hardware*, pois com uma pequena quantidade de portas lógicas pode-se construir poderosos sistemas criptográficos.

Para curva $GF(p)$ utiliza-se uma equação cúbica em que todas as variáveis e coeficientes assumem valores no conjunto de inteiros de 0 até $p - 1$ e todos os cálculos são realizados em módulo p .

Considere $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$ dois pontos na curva definida pela equação 2.8 e que a condição 2.9 seja verdadeira $\mod p$. Então $-P = (x_P, -y_P)$.

Assim tem-se $R = P + Q = (x_R, y_R)$ é determinado por:

$$\begin{cases} x_R = (\lambda^2 - x_P - x_Q) \mod p \\ y_R = (\lambda(x_P - x_R) - y_P) \mod p \end{cases} \quad (2.13)$$

onde

$$\lambda = \begin{cases} \left(\frac{y_Q - y_P}{x_Q - x_P} \right) \mod p & \text{se } P \neq Q \\ \left(\frac{3x^2 + a}{2y_P} \right) \mod p & \text{se } P = Q \end{cases} \quad (2.14)$$

2.1.9 Mapeamento Bilinear e Emparelhamento

Seja $\mathbb{G}$ e $\mathbb{G}_T$ grupos cíclicos de ordem prima r e g um gerador de $\mathbb{G}$, um mapeamento bilinear ou um emparelhamento bilinear $\hat{e}$ é uma função de dois argumentos que é linear em relação a cada uma das variáveis e que é computacionalmente eficiente [33]

$$\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$$

satisfazendo as propriedades [34]:

1. Não-degeneração: $\hat{e} : (g, g) \neq 1$;
2. Bilinearidade: $\hat{e} : (aP, bQ) = \hat{e} : (P, Q)^{ab}, \quad \forall P, Q \in \mathbb{G} \text{ e } a, b \in \mathbb{Z}_p$.

2.1.10 Emparelhamento de Tate

O emparelhamento de Weil foi usado na criptografia inicialmente por Menezes, Okamoto e Vanstone [35] no estudo do problema do logaritmo discreto em curvas elípticas. Dando

prosseguimento a este trabalho, Frey e Rück propuseram o emparelhamento de Tate [36]. Os conceitos apresentados a seguir foram extraídos de Lynn [33].

Seja E uma curva elíptica que contém n pontos sobre um corpo $\mathbb{F}_q$. Seja G um subgrupo cíclico de $E(\mathbb{F}_q)$ de ordem r , sendo que r, q são primos relativos. Seja k o menor inteiro positivo tal que $r \mid q^k - 1$.

O emparelhamento de Tate

$$\hat{e} : E[r] \cap E(\mathbb{F}_{q^k}) \times E(\mathbb{F}_{q^k}) / rE(\mathbb{F}_{q^k}) \rightarrow K^* / K^{*r}$$

é definido a seguir:

Seja f_P uma função racional com divisor $(f_P) = (P)^r$. Seja $R \in E(\mathbb{F}_q)$ tal que $R \neq P, P - Q, O, -Q$. Então, se define:

$$f(P, Q) = \frac{f_P(Q + R)}{f_P(R)} \quad (2.15)$$

2.2 Criptografia de Chave Pública

Desde sua origem até a década de 1970, os sistemas criptográficos baseavam-se em ferramentas de permutações e substituições, compartilhando um segredo comum: a chave secreta simétrica usada para cifrar e decifrar.

Whitfield Diffie e Martin E. Hellman apresentaram em 1976 o primeiro método prático de compartilhar uma chave secreta através de um canal de comunicação desprotegido. A idéia extremamente criativa baseia-se no problema do logaritmo discreto, e o algoritmo proposto recebeu o nome de Troca de Chaves Diffie Hellman [37].

Em 1978, os matemáticos do MIT (*Massachusetts Institute of Technology*), Ron Rivest, Adi Shamir e Len Adleman propuseram o primeiro esquema criptográfico de chaves públicas que ficou conhecido como RSA [22]. Desde então, o RSA é a técnica mais aceita para sistemas criptográficos de chaves públicas [25].

Apesar do sucesso dos esquemas de chaves públicas, os sistemas de chaves simétricas não ficaram obsoletos. Ao contrário, é pouco provável que os sistemas simétricos sejam abandonados, principalmente pelo custo computacional dos sistemas de chaves públicas.

2.2.1 Sistema RSA

Um dos sistemas criptográficos mais conhecidos é o RSA, cuja sigla remete aos inventores do sistema. O funcionamento do RSA fundamenta-se no fato de não ser conhecido o algoritmo capaz de fatorar rapidamente um número muito grande n que faz parte da chave pública $K_U(n, e)$ em números primos p, q tal que $n = p \cdot q$.

1. Escolher dois números primos aleatórios p e q .
2. Calcular $n = p \cdot q$.
3. Sendo $\phi(n)$ a função totiente de Euler, calcular

$$\phi(n) = (p - 1) \cdot (q - 1)$$
4. Escolher um número aleatório $e < \phi(n)$ tal que
 $\text{mdc}(e, \phi(n)) = 1$, e calcular um inteiro d de tal maneira que

$$ed \equiv 1 \pmod{\phi(n)};$$

5. Chave pública $\rightarrow K_U = (n, e)$.
 Chave privada $\rightarrow K_R = (p, q, \phi(n), d)$

6. Cifrar

Dado $m \in \mathbb{Z}_n$

$$c \equiv m^e \pmod{n}$$

7. Decifrar

$$m \equiv c^d \pmod{n}$$

Figura 2.5: Algoritmo RSA

A Figura 2.5 mostra o Algoritmo RSA¹ [38]. Nele, escolhem-se dois números primos p e q da ordem de 1024 bits para calcular um número n . Escolhe-se um número e aleatoriamente de modo que

$$e < \phi(n)$$

¹ Algoritmos simples para fins didáticos

$\phi(n)$ é a função totiente de Euler que retorna a quantidade de primos relativos de n . Como $n = p \cdot q$ e p e q são primos, todos os números menores que p são primos relativos a p e todos os números menores que q também são primos relativos a q . Assim, $\phi(n) = (p-1) \cdot (q-1)$.

Como o texto cifrado é $c = m^e \pmod n$, percebe-se que a decifragem funciona, pois

$$c^d \pmod n \equiv (m^e)^d \pmod n \equiv m^{ed} \pmod n$$

e

$$ed = k\phi(n) + 1 = k(p-1)(q-1) + 1$$

Então, se tem que:

$$m^{ed} \pmod n \equiv m^{k(p-1)(q-1)+1} \pmod n \equiv m \cdot m^{k(p-1)(q-1)} \pmod n$$

E pelo pequeno teorema de Fermat tem-se:

$$m^{k(p-1)(q-1)} \equiv m^{(p-1)(q-1)} \equiv 1 \pmod n$$

Assim:

$$c^d \equiv m \pmod n$$

2.2.1.1 A Segurança do RSA

A segurança do RSA é baseada na dificuldade de fatoração de um número n em dois números primos p e q . Existem quatro abordagens de ataque à segurança do RSA: ataque de força bruta, ataque de temporização, ataque de texto cifrado escolhido – CCA (*Chosen Ciphertext Attack*) e ataques matemáticos.

O ataque de força bruta consiste em tentar todas as combinações possíveis de chaves privadas. Para evitar este tipo de ataque, deve-se escolher um tamanho de chave grande para que a quantidade de tentativas necessárias torne o ataque impraticável.

O ataque de temporização [39] observa o tempo gasto nas operações matemáticas de decifragem e a partir destas observações descobre a informação secreta. Apesar de o ataque de temporização ser uma ameaça séria, algumas contramedidas simples podem ser usadas, como garantir um tempo de operação constante ou introduzir atrasos aleatórios.

O RSA é vulnerável ao ataque de texto cifrado escolhido. Neste tipo de ataque, o adversário escolhe diversos textos cifrados e, então, recebe o texto decifrado. Claramente, nenhuma informação nova é obtida. Ao invés disso, o adversário aproveita-se da seguinte propriedade do RSA:

$$E(K_U, M_1) \times E(K_U, M_2) = E(K_U, M_1 \times M_2) \quad (2.16)$$

onde $E(K_U, M)$ é o texto cifrado da mensagem M com a chave pública K_U .

Pode-se descobrir um texto secreto M , onde $C = M^e \pmod n$, da seguinte maneira:

1. Calcula-se $X = (C \times 2^e) \pmod n$;
2. Envia-se X como texto cifrado e recebe-se de volta $Y = X^d \pmod n$;
3. Observa-se:

$$X = (C \pmod n) \times (2^e \pmod n) = (M^e \pmod n) \times (2^e \pmod n) = (2M \pmod n);$$

4. A partir de $Y = (2M) \pmod n$ deduz-se M .

Para evitar este tipo de ataque, o RSA *Security Inc.* recomenda uma modificação no texto original não cifrado, conhecido como preenchimento ideal de criptografia assimétrica [40].

Os esforços dos pesquisadores concentram-se ultimamente em desenvolver e aprimorar algoritmos que permitam a descoberta de p e q . São chamados de **ataques matemáticos**, pois exploram características matemáticas no RSA.

O aumento do poder de processamento e o refinamento dos algoritmos de fatoração ameaçam continuamente o RSA. A Tabela 2.1 mostra a evolução dos resultados alcançados pelos pesquisadores na fatoração de chaves do RSA.

Em 2007, pesquisadores anunciaram a fatoração de um número especial composto:

$$\left(\frac{2^{1039} - 1}{5080711} \right)$$

de aproximadamente 1017 bits, com esforço computacional equivalente à quebra de um número de 700 bits [41]. Tal resultado lança dúvidas sobre a longevidade de chaves de 1024 bits.

Tabela 2.1: Evolução da Fatoração

Dígitos Decimais	Número de bits	Ano	Algoritmo
110	332	1991	Crivo quadrático
110	365	1992	Crivo quadrático
120	398	1993	Crivo quadrático
129	428	1994	Crivo quadrático
130	431	1996	Crivo de corpo numérico generalizado
140	465	1999	Crivo de corpo numérico generalizado
155	512	1999	Crivo de corpo numérico generalizado
160	530	2003	Crivo de malha(<i>Lattice sieve</i>)
174	576	2003	Crivo de malha(<i>Lattice sieve</i>)
200	663	2005	Crivo de malha(<i>Lattice sieve</i>)

Fonte: Stallings [25]

A necessidade de chaves cada vez maiores preocupa em função *overhead* computacional, principalmente pela crescente presença de dispositivos móveis com baixo poder de processamento que necessitam usar sistemas criptográficos. Alternativas como o ECC começam a despertar o interesse da comunidade acadêmica e empresarial.

2.2.2 Criptografia de Curva Elíptica

O uso das curvas elípticas em $\mathbb{R}$ não é prático tampouco preciso. Aplicações criptográficas precisam de uma aritmética rápida e precisa, o que pode ser obtido com o uso de corpos inteiros finitos [26, 31]. Agora, ao invés de uma curva, tem-se um conjunto de pontos $E_p(a, b)$ que satisfaz a equação $y^2 = x^3 + ax + b \pmod{p}$.

Para formar um sistema criptográfico usando curvas elípticas, deve-se encontrar um sistema difícil de resolver, assim como a fatoração de produto de dois números primos ou encontrar o logaritmo discreto.

Considere a equação $R = kS$, onde $R, S \in E_p(a, b)$ e $k < p$. É fácil calcular R a partir de k, S , porém é difícil encontrar k a partir de R e S [25]. Conhecendo k pode-se, por exemplo, encontrar $S = kP$ (para $k = 100$), da seguinte maneira:

$$S = P + P + 2P + 4P + 8P + 16P + 32P + 32P + 4P$$

Para encontrar k a partir de R e S , deve-se tomar $k = 1, 2, 3 \dots n$ até encontrar-se $kP = S$.

Esta dificuldade matemática pode ser usada em um sistema criptográfico. Uma abordagem simples de um sistema criptográfico com curvas elípticas é descrito a seguir.

Uma mensagem m que se deseja transmitir pode ser traduzida em um ponto x, y P_m . Este ponto P_m será cifrado e posteriormente decifrado. Como nem toda mensagem pode ser traduzida em pontos de $E_p(a, b)$ (não existem todas as coordenadas em $E_p(a, b)$), deve-se escolher um ponto G e um grupo elíptico $E_p(a, b)$ que formarão os parâmetros do sistema.

Alice escolhe um número de chave privada $K_{RA} = n_A$ e calcula sua chave pública $K_{UA} = n_A G$. Para Bob enviar uma mensagem P_m para Alice, escolhe um número inteiro k e produz um texto cifrado C_m que consiste de dois pontos

$$C_m = \{kG, P_m + kK_{UA}\}$$

Para Alice decifrar a mensagem enviada por Bob, ela multiplica o primeiro ponto pela sua chave privada e subtrai o resultado do segundo ponto

$$P_m + kK_{UA} - n_A kG = P_m + k(n_A G) - n_A kG = P_m$$

2.2.2.1 Segurança da Criptografia de Curvas Elípticas

Determinar k , dados kP e P , é conhecido como problema do logaritmo elíptico. O método mais conhecido para o cálculo do logaritmo elíptico é o Pollard Rho [25].

Tabela 2.2: Comparação do Tamanho de Chaves em Termos de Esforço Computacional

Simétrico	ECC	RSA/DSA
(tamanho da chave em bits)	(tamanho da chave em bits)	(tamanho da chave em bits)
80	160	1024
112	224	2048
128	256	3072
192	384	8192
256	512	15360

Fonte: Hankerson, Menezes e Vanstone [30]

A Tabela 2.2 mostra tamanho de chaves comparáveis em termos de esforço computacional para criptoanálise. Constata-se que o ECC pode usar um tamanho de chave consideravelmente menor que o RSA com um nível de segurança equivalente. Além disso, para tamanhos de chaves

iguais, os custos computacionais demandados pelo ECC e pelo RSA são de mesma ordem de grandeza [42].

Tabela 2.3: Avanços na Resolução do Problema do Logaritmo Elíptico

ECC (Tamanho da chave em bits)	Data da Resolução
79	1997
89	1998
97	1999
109	2004

Fonte: CERTICOM [43]

A Tabela 2.3 [43] ilustra os avanços na resolução do problema do logaritmo elíptico. Nota-se que a maior chave com solução possui um valor cujo tamanho análogo de chave RSA é de aproximadamente 485 bits.

2.2.3 Funções de *Hash*

Uma função de *hash* recebe uma mensagem M de tamanho variável e produz uma saída $H(M)$ de tamanho fixo, conhecida como resumo da mensagem ou valor de *hash*. O código de *hash* é uma função de todos os bits de M e qualquer mudança na mensagem resulta em uma alteração grande no valor de $H(M)$. Na Tabela 2.4 tem-se o valor de *hash* de uma cadeia de caracteres. Observa-se que trocando uma única letra, o código de *hash* tem uma grande alteração. Este efeito avalanche mostra-se desejável para evitar que adversários tentem encontrar um valor de *hash* específico fazendo pequenas alterações na mensagem original.

Tabela 2.4: Valor de *hash* SHA-1 de uma Cadeia de Caracteres

Palavra	Hash
teste	9dc628289966d144c1a5fa20dd60b1ca1b9de6ed
testa	3d4ea5604892b4a48168bd26026a737990ac5d7b

Uma função *hash* tem como finalidade produzir uma “impressão digital” da mensagem M que pode ser útil para verificação da integridade ou para autenticação de M . Como o valor de *hash* possui uma saída de valor fixo, tem-se uma função de **muitos-para-um**, ou seja, várias

mensagens podem ter o mesmo valor de *hash*. Para evitar ataques, é interessante que uma função *hash* H tenha as seguintes propriedades [25]:

1. H pode ser aplicado a um bloco de dados de qualquer tamanho;
2. H produz uma saída de comprimento fixo;
3. $H(x)$ é relativamente fácil de calcular para qualquer x ;
4. Para qualquer valor de $H(x)$ dado, é computacionalmente inviável encontrar o valor x . Isso é conhecido como **propriedade unidirecional**;
5. Para qualquer bloco x , é computacionalmente inviável encontrar $y \neq x$ tal que $H(y) = H(x)$;
6. É computacionalmente inviável encontrar um par (x, y) tal que $H(y) = H(x)$.

As três primeiras propriedades são requisitos para a aplicação prática de funções *hash* para autenticação. A propriedade 4 garante que seja fácil calcular o código *hash* de uma mensagem, mas impossível calcular a mensagem a partir de seu código. Essa propriedade é importante, caso a transmissão envolva o envio de um valor secreto. Sem essa propriedade, um adversário pode descobrir o valor secreto a partir do valor de *hash*.

A propriedade 5 garante que é muito difícil encontrar outra mensagem y que produza um valor de *hash* igual ao de uma dada mensagem x . Na ausência desta propriedade, um adversário pode falsificar uma mensagem de maneira que a mensagem forjada tenha o mesmo *hash* da mensagem original.

A propriedade 6 está relacionada com resistência da função *hash* ao ataque popularmente conhecido como ataque do aniversário. Isso significa que, dada uma função *hash* H , que produza uma saída de n bits, existem 2^n valores de saída possíveis. No ataque do aniversário, se H é usada em um número K de mensagens aleatórias, o valor de K para que a probabilidade de que pelo menos uma entrada y satisfaça $H(y) = H(x)$ seja igual a $\frac{1}{2}$ é $2^{\frac{n}{2}}$.

O ataque do aniversário baseia-se no paradoxo do aniversário, segundo o qual o valor mínimo N de um grupo de pessoas, para que a probabilidade de duas pessoas fazerem aniversário na mesma data seja de $\frac{1}{2}$ é $2^{\frac{n}{2}}$.

Apesar da popularidade do algoritmo de resumo de mensagem MD5 (*Message Digest 5*), sua saída de 128 bits é considerada vulnerável a ataques de força bruta e criptoanálise. Como substituto surgiu o SHA (*Secure Hash Algorithm*)

2.2.3.1 Algoritmo de *Hash* Seguro - SHA

O SHA foi desenvolvido pelo NIST (*National Institute of Standards and Technology*) em 1993 (FIPS-180). Em 1995 foi revisado como FIPS 180-1 e é conhecido como SHA-1. O SHA-1 produz um valor de hash de 160 bits baseado na função de *hash* MD4. No ano de 2002, o NIST publicou o FIPS 180-2, uma revisão do SHA, no qual definiu três novas versões com valores de *hash* de tamanhos 256, 384 e 512 bits. Em 2008 foi publicada a versão final no FIPS-180-3 [44].

Em 2005 uma pesquisa mostrou um ataque em que um mesmo valor de hash poderia ser encontrado para duas mensagens diferentes usando 2^{69} operações, abaixo do valor de 2^{80} necessários anteriormente para encontrar uma colisão de valores de hash [45].

O algoritmo SHA256 aceita uma entrada de tamanho menor que 2^{64} bits e tem uma saída de 256 bits. A entrada é processada em blocos de 512 bits. Já o algoritmo SHA512 aceita uma entrada menor que 2^{128} bits e produz uma saída de 512 bits. A entrada é processada em blocos de 1024 bits. A segurança do SHA pode ser calculada por $2^{\frac{n}{2}}$, sendo que n é o número de bits da saída do SHA e $2^{\frac{n}{2}}$ é a quantidade de tentativas necessárias para o ataque do aniversário produzir uma colisão.

2.3 Assinatura Baseada em Identidade

Um dos avanços mais importantes proporcionados pela criptografia de chaves públicas é a assinatura digital. Vários recursos de segurança não seriam possíveis de se desenvolver sem o uso da assinatura digital. Por exemplo, a confidencialidade protege a comunicação entre duas partes de uma terceira, porém não protege as duas partes uma da outra.

Um possível cenário de ataque pode ser descrito assim: Alice e Bob possuem um canal seguro para troca de mensagens. Este canal seguro utiliza uma chave secreta comum a Alice e Bob. Alice pode forjar uma mensagem e utilizando a chave secreta alegar que ela veio de Bob.

Outro cenário possível ocorre quando Bob pode enviar uma mensagem para Alice e depois negar o envio. Uma vez que é possível para Alice forjar uma mensagem, não é possível provar que Bob realmente não a enviou.

É possível utilizar a criptografia de chaves públicas para produzir assinaturas digitais. A Figura 2.6 esquematiza Bob produzindo uma mensagem e cifrando-a com sua chave privada. Como todos possuem acesso à chave pública de Bob, todos podem decifrar a mensagem cifrada por ele. Já a chave secreta é conhecida somente por Bob. Portanto, as pessoas que decifraram

a mensagem com a chave pública terão a certeza de que somente Bob poderia ter produzido o documento, ou seja, o documento foi assinado por Bob.

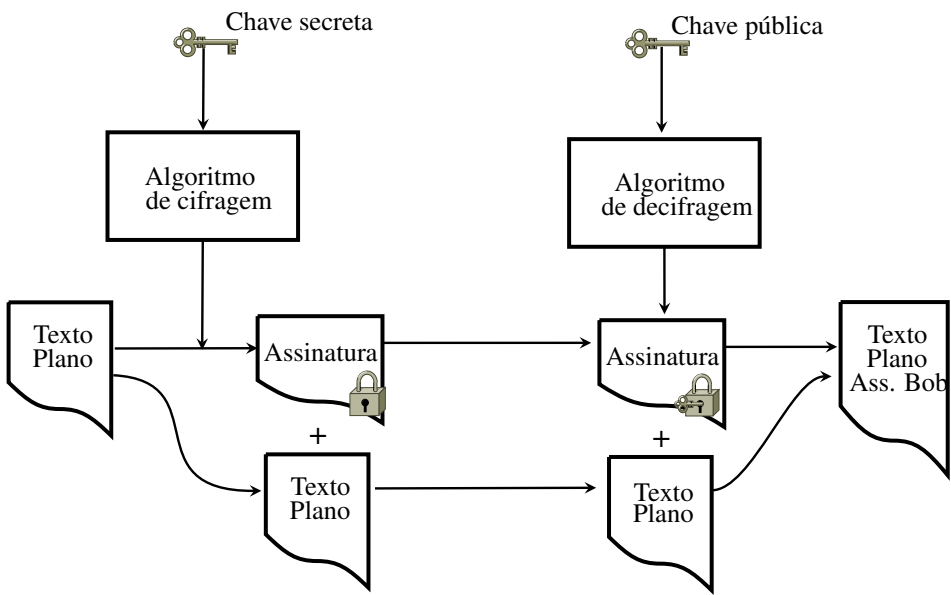


Figura 2.6: Esquema de Assinatura Digital

Para que o esquema de assinatura digital funcione, é imprescindível que a chave pública do signatário da mensagem seja conhecida por todos, o que nem sempre é possível. A Tabela 2.5 mostra uma chave pública de 1024 bits codificada no formato base64. É improvável que usuários consigam memorizar e associar a uma pessoa, um número decimal de mais de 300 dígitos ou uma *string* de mais de 200 caracteres codificados no formato base64.

Tabela 2.5: Chave Pública RSA

Algoritmo	Tamanho em bits	Chave codificação base64
RSA	1024	MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDuZpWX9tQkBPasNtI468StDjHgXFPtNAwL73tHzGUWULSJNx2xTHWeMIO2HqD9CE5vW/6QVB0c+S5x4ULPNwH48bOwf7HOX66MjMCj9Wf4IO4UTVMwUXUs0LzbdoyUkZaoRMFgxdqlByKeEVOwh66Uzm+f nHJv2I/6LnAvLaEy0wIDAQAB

Uma situação, em que um adversário forja e assina uma mensagem com uma chave própria não pode ser descoberta sem o auxílio de uma terceira parte confiável. Esta entidade confiável, também chamada de árbitro, é responsável por garantir a identidade do usuário, dono de uma chave. Usualmente, o árbitro emite um certificado assinado relacionando a chave à identidade do usuário. Neste caso, o interessado em verificar a assinatura deve conhecer a chave pública do árbitro e consultar uma lista de certificados revogados.

Em 1984, Adi Shamir comentou que seria mais eficiente se não houvesse a necessidade de associar entidades às chaves por meio de certificados [46]. As entidades poderiam utilizar as próprias identidades como chave pública. Assim, a chave pública poderia ser determinada facilmente, desde que a identidade da entidade fosse conhecida.

A Tabela 2.6 mostra exemplos de chaves públicas. Observa-se que a chave pública no IBS é mais simples e a própria chave indica a quem ela pertence e em qual data a validade da chave expira.

Tabela 2.6: Chaves Públicas RSAxIBS

Algoritmo	Chave
RSA	MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDuzpWX9tQkBPasNtl468StDjHgXFPtNAwL73tHzGUWULSJN2xTHWeMIO2HqD9CE5vW/6QVB0c+S5x4ULPNwH48bOwf7HOX66MjMCj9Wf4IO4UTVMwUXUs0LzbdoYUkZaoRMFgxdqlByKeEVOwh66Uzm+fnHJv2I/6LnAvLaEy0wIDAQAB
IBS	alice@domain.com.br20081220

A assinatura de uma mensagem com algoritmos de chave pública tem o inconveniente de produzir uma assinatura com tamanho de mesma ordem de grandeza do documento a ser assinado. Este fato torna-se um inconveniente tanto para armazenar como para transmitir a assinatura.

Para contornar este problema, utiliza-se uma função *hash* para criar uma “impressão digital” de tamanho fixo da mensagem e o valor de *hash* é que será assinado.

Criar uma chave privada a partir de uma chave pública (identidade) parece contraditório com a idéia de que deve ser matematicamente muito difícil encontrar a chave privada a partir da chave pública, porém alguns trabalhos [34, 46–48] apresentaram soluções inteligentes para o problema.

2.3.1 Esquema de Assinatura Baseada em Identidade de Paterson

Paterson propôs um esquema de IBS cujos objetivos foram eficiência computacional e segurança [48]. Este esquema é similar ao esquema de assinatura de ElGamal [49].

Seja G_1 um grupo aditivo de ordem prima q e G_2 um grupo multiplicativo de mesma ordem q . Assume-se que existe um mapeamento bilinear $\hat{e}$ de $G_1 \times G_1$ para G_2 . Também assume-se um elemento $P \in G_1$, satisfazendo $\hat{e}(P, P) \neq 1_{G_2}$, que é conhecido como parâmetro do sistema.

Seja ID a cadeia de caracteres usada para representar a identidade do signatário da mensagem M e H_1, H_2 e H_3 funções *hash* públicas, requer-se que $H_1 : \{0, 1\}^* \rightarrow G_1$, $H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ e $H_3 : G_1 \rightarrow \mathbb{Z}_q$.

Apresenta-se a seguir o algoritmo de assinatura e verificação proposto por Paterson [48] em 2002:

1. Seja $ID \rightarrow$ a cadeia de caracteres associada à identidade do signatário;
2. Seja um elemento $P \in G_1$ conhecido, satisfazendo $\hat{e}(P, P) \neq 1_{G_2}$;
3. Calcula-se $Q_{ID} = H_1(ID)$ para ser usado na verificação da assinatura;
4. A chave secreta para assinar a mensagem é dada por $D_{ID} = s \cdot Q_{ID}$, onde $s \in \mathbb{Z}_q$.
5. Considera-se também que o parâmetro $P_{PUB} = s \cdot P$ é conhecido de maneira pública;
6. Para assinar uma mensagem M , um usuário escolhe um valor aleatório $k \in \mathbb{Z}_q^*$, calcula a assinatura como um par $(R, S) \in G_1 \times G_1$, onde:

$$R = k \cdot P$$

$$S = k^{-1}(H_2(M) \cdot P + H_3(R) \cdot D_{ID})$$

Aqui, k^{-1} é o inverso de k em $\mathbb{Z}_q^*$.

7. Para verificar uma assinatura (U, V) de uma mensagem M , calcula-se $\hat{e}(U, V)$;
8. A assinatura é verificada se:

$$\hat{e}(U, V) = \hat{e}(P, P)^{H_2(M)} \cdot \hat{e}(P_{PUB}, Q_{ID})^{H_3(R)}$$

Figura 2.7: Algoritmo IBS de Paterson

Usando as propriedades de bilinearidade em $\hat{e}$, verifica-se que o Algoritmo da Figura 2.7 funciona pois:

$$\begin{aligned}
\hat{e}(R, S) &= \hat{e}(k \cdot P, k^{-1}(H_2(M) \cdot P + H_3(R) \cdot D_{ID})) \\
&= \hat{e}(P, H_2(M) \cdot P + H_3(R) \cdot D_{ID})^{k^{-1} \cdot k} \\
&= \hat{e}(P, H_2(M) \cdot P + H_3(R) \cdot D_{ID}) \\
&= \hat{e}(P, H_2(M)P) \cdot \hat{e}(P, H_3(R) \cdot s \cdot Q_{ID}) \\
&= \hat{e}(P, P)^{H_2(M)} \cdot \hat{e}(P, Q_{ID})^{s \cdot H_3(R)} \\
&= \hat{e}(P, P)^{H_2(M)} \cdot \hat{e}(s \cdot P, Q_{ID})^{H_3(R)} \\
&= \hat{e}(P, P)^{H_2(M)} \cdot \hat{e}(P_{PUB}, Q_{ID})^{H_3(R)}
\end{aligned}$$

2.3.2 Esquema de Assinatura Baseada em Identidade de Hess

Em 2003, Florian Hess propôs um esquema de IBS [50] melhorando a eficiência em relação ao esquema de Paterson [48]. O esquema IBS de Hess é descrito a seguir:

Seja G_1 um grupo cíclico aditivo de ordem prima q e G_2 um grupo multiplicativo de mesma ordem q , assume-se que existe um mapeamento bilinear $\hat{e}$ de $G_1 \times G_1$ para G_2 . Seja $P \in G_1$ um gerador de G_1 , definem-se as seguintes funções de *hash*:

$$\begin{aligned}
H_1 : \{0, 1\}^* \times G_2 &\rightarrow (\mathbb{Z}/q\mathbb{Z})^\times \\
H_2 : \{0, 1\}^* &\rightarrow G_1^*
\end{aligned}$$

-
1. Seja $ID \rightarrow$ a cadeia de caracteres associada à identidade do signatário;
 2. Seja s um valor aleatório onde $s \in (\mathbb{Z}/q\mathbb{Z})^\times$, calcula-se $P_{PUB} = sP$ e publica-se P_{PUB} enquanto mantém-se s secreto.
 3. A chave secreta para assinar a mensagem é dada por $D_{ID} = s \cdot H_2(ID)$.

4. Para assinar uma mensagem M o signatário escolhe um $P_1 \in G^*$ arbitrário e pega um inteiro aleatório $k \in (\mathbb{Z}/q\mathbb{Z})^\times$ e calcula:

$$(a) \quad r = \hat{e}(P_1, P)^k;$$

$$(b) \quad v = H_1(M, r);$$

$$(c) \quad u = vD_{ID} + kP_1.$$

A assinatura é o par (u, v) .

5. Ao receber uma mensagem M com uma assinatura (u, v) o destinatário calcula:

$$(a) \quad r = \hat{e}(u, P) \cdot \hat{e}(H_2(ID), -P_{PUB})^v;$$

$$(b) \quad \text{A assinatura é válida se e somente se } v = H_1(M, r).$$

Figura 2.8: Algoritmo IBS de Hess

Usando as propriedades de bilinearidade em $\hat{e}$, verifica-se que o Algoritmo da Figura 2.8 funciona pois:

$$\begin{aligned} r &= \hat{e}(u, P) \cdot \hat{e}(H_2(ID), -P_{PUB})^v \\ &= \hat{e}(vD_{ID} + kP_1, P) \cdot \hat{e}(H_2(ID), -P_{PUB})^v \\ &= \hat{e}(vH_2(ID), P) \cdot \hat{e}(kP_1, P) \cdot \hat{e}(H_2(ID), -P_{PUB})^v \\ &= \hat{e}(vH_2(ID), P)^s \cdot \hat{e}(H_2(ID), sP)^{-v} \cdot \hat{e}(P_1, P)^k \\ &= \hat{e}(vH_2(ID), P)^s \cdot \hat{e}(H_2(ID), P)^{-sv} \cdot \hat{e}(P_1, P)^k \\ &= \hat{e}(vH_2(ID), P)^s \cdot \hat{e}(vH_2(ID), P)^{-s} \cdot \hat{e}(P_1, P)^k \\ &= \hat{e}(P_1, P)^k \end{aligned}$$

2.4 Autenticação de Remetente

A assinatura digital baseada na identidade juntamente com o serviço de DNS (*Domain Name System*) e a aplicação de correio eletrônico, dão suporte para assinar e verificar a identidade do remetente. O DNS permite a publicação e consulta de parâmetros públicos definidos pelo CA (*Certificate Authority*). A Figura 2.9 mostra uma abstração de autenticação de remetente, utilizado no trabalho proposto nesta dissertação, e apresentado no Capítulo 4.

A ilustração é uma abstração de todo o processo, usando um sistema baseado na identidade para autenticação do remetente de uma mensagem de Correio Eletrônico. Há basicamente quatro níveis de abstração: o aplicativo de Correio Eletrônico, a IBS (Assinatura Baseada na Identidade), a CA (Autoridade Certificadora) e o DNS. Dentro de cada um destes níveis pode-se abstrair métodos específicos de ação.

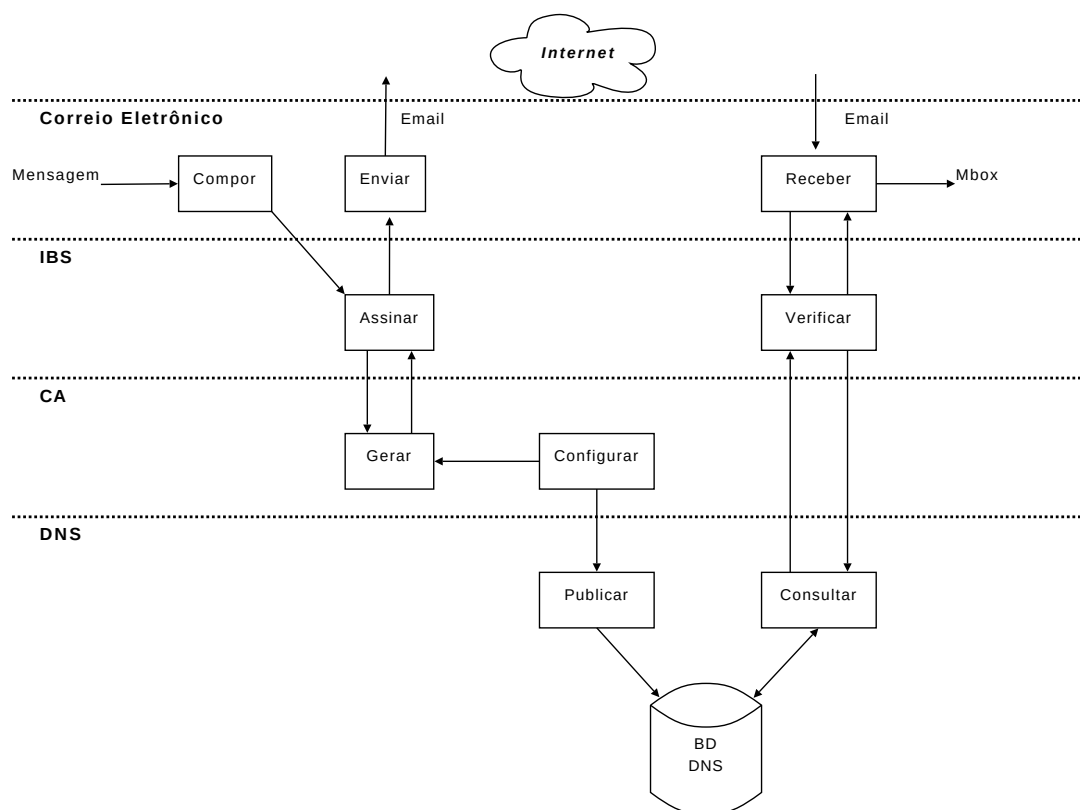


Figura 2.9: Abstração do Processo de Autenticação de Remetente

No nível de abstração relativo ao Correio Eletrônico, há o método compor que irá criar a mensagem de e-mail para que seja enviada ao destinatário via *Internet*. Há também o método receber que entregará à caixa de correio correspondente ao destinatário, *Mbox*, o e-mail proveniente da *Internet*.

O nível de IBS possui o método Assinar que irá marcar com uma assinatura do remetente a mensagem a ser enviada pelo correio eletrônico. O nível IBS também possuirá um método Verificar que irá validar as assinaturas das mensagens, usando parâmetros públicos associados ao CA.

No nível de abstração CA, existe o método Configurar onde parâmetros públicos e um parâmetro secreto são criados. O parâmetro secreto será usado pelo método Gerar para a produção de Chaves Privadas, requisitadas pelo método Assinar, associadas às Chaves Públicas dos usuários.

O nível de DNS tem um método Publicar que ao receber os parâmetros públicos do método Configurar, armazena-os no Banco de Dados do DNS. As informações, armazenadas no Banco de Dados do DNS, são usadas pelo método Consultar que informa ao método Verificar os parâmetros públicos.

Neste capítulo, foram abordados conceitos segurança de dados, como criptografia de chaves públicas, assinatura digital e assinatura digital baseada em identidade. Também foi abordado os conceitos matemáticos necessários para o entendimento dos conceitos de segurança de dados. O capítulo a seguir aborda conceitos relacionados ao correio eletrônico, como protocolos, aplicações e filtros.

Capítulo 3

Correio Eletrônico

O serviço de mensagem eletrônica remonta ao surgimento da *Internet*, tendo sido uma das aplicações mais populares nos seus primórdios [Segaller [51], 1998 apud Kurose e Ross [17], 2006], tornando-se cada vez mais sofisticado com o passar do tempo e continua em transformação.

Dentre as conhecidas vantagens do e-mail em relação ao correio convencional destaca-se o fato de o *e-mail* trazer novos recursos, como a possibilidade de anexar arquivos, conteúdo em hipertexto, conteúdo multimídia entre outros. Embora grande parte das mensagens seja de texto, o fato de se anexar arquivos, possibilita que o e-mail seja usado como um meio de transmissão assíncrona de voz e vídeo.

Neste capítulo são apresentados fundamentos teóricos necessários para a compreensão dos conceitos relacionados ao correio eletrônico e ao protocolo SMTP

3.1 Protocolo SMTP

O SMTP é o principal protocolo da camada de aplicação do correio eletrônico da *Internet*. Devido às suas características de tolerância a atrasos e intolerância a perdas, o SMTP utiliza o protocolo TCP (*Transmission Control Protocol*), que é um protocolo da camada de transporte com transmissão confiável de dados.

O protocolo SMTP foi definido inicialmente pela RFC876 [2], atualizado pela RFC2821 [3] e posteriormente pela RFC5321 [4]. Fundamentado na arquitetura cliente-servidor, o protocolo SMTP transfere mensagens de servidores remetentes para servidores destinatários.

A Figura 3.1 esquematiza o projeto do SMTP. O usuário, por meio de um MUA (*Mail*

User Agent), envia uma ou mais mensagens para um ou vários destinatários por meio do cliente SMTP. Cabe ao cliente transferir para um ou mais servidores SMTP¹ as mensagens do usuário. Caso a transferência não seja possível, o cliente deve retornar um aviso de falha.

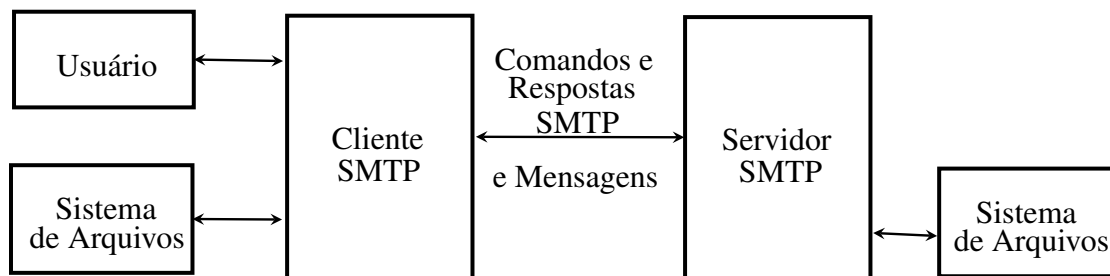


Figura 3.1: Esquema Básico do Projeto do SMTP

Se o cliente encontrar o servidor do destinatário, mas não conseguir entregar a mensagem por algum motivo, o cliente armazenará temporariamente a mensagem no seu sistema de arquivos para tentar transmiti-la posteriormente.

O servidor, ao receber a transmissão do cliente, armazena a mensagem em seu sistema de arquivos até o destinatário requisitá-la. Eventualmente, o servidor pode interpretar que a mensagem recebida deve ser repassada a outro servidor. Neste caso, ao repassar a mensagem, passa a atuar como cliente.

3.1.1 Iniciação de Seção SMTP

Uma seção é iniciada quando um cliente abre uma conexão com um servidor e este responde com uma mensagem de abertura. Algumas aplicações exibem na mensagem de abertura, informações do *software* e da versão. Há administradores que preferem omitir tais informações para dificultar que adversários obtenham informações que afetem a segurança dos servidores.

O servidor deve responder que a iniciação foi feita com sucesso. Em caso de problemas, o servidor deve informar e aguardar que o cliente encerre a seção, com dados suficientes para o cliente rastrear a origem do problema. Uma vez que o servidor enviou a mensagem de boas vindas e o cliente a recebeu, o cliente envia um comando **EHLO** juntamente com sua identidade. Com o comando **EHLO** o cliente também indica que suporta extensões de serviços.

A Figura 3.2 exibe uma impressão de tela com uma inicialização de seção SMTP, na qual o cliente envia o comando **EHLO** e o servidor responde com as extensões de serviços

¹Tanto o cliente como o servidor SMTP podem ser denominados MTA (*Mail Transfer Agent*)

disponíveis. Uma vez concluída a inicialização, o protocolo passa para a fase de transações de mensagens.

```
root@note:~# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 note. ESMTP Sendmail 8.14.2/8.14.2; Tue, 20 Dec 2008 00:57:36 -0200
ehlo localhost
250-note. Hello localhost [127.0.0.1], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-AUTH DIGEST-MD5 CRAM-MD5
250-DELIVERBY
250 HELP
```

Figura 3.2: Iniciação de seção SMTP

3.1.2 Transações de Mensagens

As transações de mensagens compreendem três fases. A primeira fase inicia-se com o comando MAIL em que o remetente informa sua identidade. Na segunda fase, um ou mais comandos RCPT são usados para informar os endereços dos destinatários. Na fase final, o comando DATA inicia a transferência dos dados da mensagem até que um comando de fim de transmissões de dados seja enviado.

Após a mensagem de saudação, o cliente deve começar com o comando:

```
MAIL FROM:<reverse-path> [SP<mail-parameters>] <CRLF>
```

Na parte <reverse-path> o cliente deve informar o endereço do remetente entre “<” e “>”. O servidor verifica no endereço fornecido, a existência de um domínio e a sua validade. Se o endereço do remetente for aceito, o servidor responde com 250 OK. Em <mail parameters> informam-se, caso necessário, parâmetros relacionados aos serviços estendidos. O comando é finalizado com o envio dos caracteres <CR> *carriedge-return* e <LF> *line-feed* que, juntos, indicam o final do comando.

Na segunda fase da transação deve-se enviar o comando:

```
RCPT TO:<forward-path> [SP<rcpt-parameters>] <CRLF>
```

Em <forward-path> informa-se o endereço do destinatário. Este endereço receberá as mesmas verificações que o endereço do remetente recebe. Se o endereço do destinatário for aceito, o servidor responde com 250 OK. Caso contrário, o servidor responde 550 seguido de uma mensagem explicativa.

A terceira e última fase é iniciada com o comando:

```
DATA <CRLF>
```

Se o comando DATA for aceito, o servidor responde com 354 juntamente com o caractere que indica o final do envio de dados da mensagem. A Figura 3.3 mostra uma impressão da tela com os comandos e respostas de uma transação de mensagem. Na transação da Figura 3.3, o caractere “.” em uma linha vazia encerra a transmissão de dados.

```
mail from:<root@localhost>
250 2.1.0 <root@localhost>... Sender ok
rcpt to:<root@localhost>
250 2.1.5 <root@localhost>... Recipient ok
data
354 Enter mail, end with "." on a line by itself
```

Figura 3.3: Transação de Mensagens do Protocolo SMTP

O SMTP usa conexões persistentes. Assim, caso o cliente deseje enviar outra mensagem, não há necessidade de se estabelecer uma nova conexão. O servidor, portanto, não encerra a conexão sem um comando específico do cliente; ele deve transmitir um comando QUIT.

Eventualmente, o servidor SMTP pode ser um intermediário na transmissão da mensagem, tendo como função repassar ou enviar a mensagem para outro servidor. Algumas situações podem exigir a mudança do endereço do destinatário, como no caso de listas de discussão ou *aliases*, porém o protocolo SMTP não permite a mudança do campo TO do cabeçalho. Nas situações de mudança de destinatário, muda-se somente o endereço do destino RCPT TO do envelope, mantendo o destinatário do cabeçalho intacto. Ressaltar este comportamento é importante neste trabalho, pois a mudança dos campos From e To do cabeçalho pelo MTA invalidariam a assinatura digital da mensagem.

Uma limitação imposta pelo protocolo SMTP é que cada mensagem, inclusive o corpo, esteja no formato ASCII (*American Standard Code for Information Interchange*) de 7 bits. Caso a mensagem esteja em um formato diferente, por exemplo, um formato binário, ela terá que ser codificada em ASCII de 7 bits.

3.1.3 Formato da Mensagem

Uma mensagem é composta de um envelope que contém os comandos do protocolo SMTP e o conteúdo da mensagem, conforme esquematizado na Figura 1.2. O conteúdo da mensagem é dividido em duas partes: o **cabeçalho** e o **corpo**.

O cabeçalho contém algumas informações periféricas definidas na RFC5321. Dentre elas, duas são obrigatórias: uma linha contendo `From` e outra linha contendo `To`. Opcionalmente, pode-se incluir também uma linha `Subject`. Um cabeçalho de mensagem típico é semelhante a:

```
From: alice@facom.ufu.br  
To: bob@ufu.br  
Subject: Agenda do congresso.
```

Após o cabeçalho, uma linha em branco indica a separação entre o cabeçalho e o corpo da mensagem. O corpo da mensagem também deve ter suas informações codificadas no formato ASCII de 7 bits. Esta codificação restringe muito os conteúdos transmitidos atualmente, como voz e imagens.

Para enviar uma mensagem com conteúdo que não segue o padrão ASCII, cabeçalhos extras devem ser incluídos na mensagem. Estes cabeçalhos são definidos na RFC2045 [52] e na RFC2046 [53] e são referentes ao MIME (*Multipurpose Internet Mail Extensions*).

Os dois principais cabeçalhos do MIME para suporte de multimídia são `Content Type` e `Content Transfer Encoding`. O cabeçalho `Content Type` permite que o destinatário realize uma ação adequada sobre a mensagem, enquanto o campo `Content Transfer Encoding` informa qual o tipo de codificação deve ser usada para decodificar a mensagem do formato ASCII para o formato original. A Figura 3.4 mostra uma impressão de tela com o conteúdo de uma mensagem com uma imagem em anexo.

```
From: alice@note.  
Date: Tue, 30 Dec 2008 22:05:42 -0200  
To: root@note.  
Subject: Logomarca  
Message-ID: <20081231000542.GA7623@note>  
Mime-Version: 1.0  
Content-Type: multipart/mixed; boundary="a8Wt8ulKmwUX3Y2C"  
Content-Disposition: inline  
User-Agent: Mutt/1.4.2.3i
```

```
--a8Wt8ulKmwUX3Y2C  
Content-Type: text/plain; charset=us-ascii  
Content-Disposition: inline
```

```
Logomarca em anexo  
Alice
```

```
--a8Wt8ulKmwUX3Y2C  
Content-Type: image/png  
Content-Disposition: attachment; filename="brush.png"  
Content-Transfer-Encoding: base64
```

```
iVBORw0KGgoAAAANSUhEUgAAABYAAAAAPCAIAAAABvD6g/AAAAA3NCSVQICAjb4U/gAAAACXBI  
WXMAAAAAAAAAAAHqZRakAAAAAP0lEQVQokWP8//8/A2WaiUL9NDOckZERKxsXYBysYcHIyEiM  
+xHqifQIIyN0LYM2LKhgBHJYfv/+naARwyYsABc6FP8kpwY5AAAAAE1FTkSuQmCC
```

```
--a8Wt8ulKmwUX3Y2C--
```

Figura 3.4: Conteúdo de Mensagem com Imagem Anexada

O protocolo SMTP é utilizado nos servidores em aplicações MTA, sendo o *Sendmail* uma solução madura bastante popular. O *Sendmail* é caracterizado como sendo aplicação de uso geral, flexível e configurável para atender necessidades de roteamento e entrega de mensagens para todos os sítios, pequenos ou grandes, simples ou complexos. Outras opções para aplicações MTA são o Postfix [54], Qmail [55], Exim [56], MS Exchange [57] entre outros.

3.2 Sendmail

O *Sendmail* foi desenvolvido por Eric Allman enquanto estudante e membro da equipe da *University of California at Berkeley*. Uma das máquinas do campus que estava conectada na ARPAnet², era o servidor do projeto no qual Eric Allman trabalhava, que por sua vez estava conectado a outras máquinas do campus por meio de uma rede de baixo custo criada por Eric Schmidt, chamada BerkNet [58].

As mensagens eletrônicas trafegavam nas próprias redes, mas não trafegavam entre as redes. Vislumbrando um crescimento explosivo no número de redes, Eric Allman resolveu escrever um programa chamado *delivermail*. Este precursor do *Sendmail* teve problemas, pois não possuía a flexibilidade necessária para atender as mudanças que ocorriam nos roteamentos de mensagens, uma vez que sua configuração era compilada junto ao programa.

Diversas mudanças foram acontecendo como, a migração do protocolo NCP (*Network Control Protocol*) para o TCP e a mudança da transmissão de mensagens do FTP (*File Transfer Protocol*) para o SMTP [58].

Em resposta a estas e outras mudanças, Eric Allman incrementou o programa *delivermail* para o *Sendmail*. A primeira versão do programa foi distribuída na versão 4.1c do BSD (*Berkeley Software Distribution*) e, desde então, Eric Allman juntamente com outros colaboradores, vem desenvolvendo o *Sendmail* [59].

3.2.1 As Funções do Sendmail

O *Sendmail* executa uma variedade de funções na entrega das mensagens eletrônicas. Ele aguarda por novas mensagens, encaminha mensagens para outros servidores ou as repassa para programas adequados para a entrega de mensagens localmente. O *Sendmail* pode anexar as mensagens a arquivos ou redirecionar a outros programas [60]. Ele pode armazenar para entregar posteriormente e também pode interpretar apelidos que relacionam um endereço a outro.

A Figura 3.5 esquematiza a hierarquia do sistema de arquivos, que pode ser vista como uma árvore invertida, sendo que o arquivo *sendmail.cf* encontra-se na raiz da árvore.

²Rede precursora da *Internet*

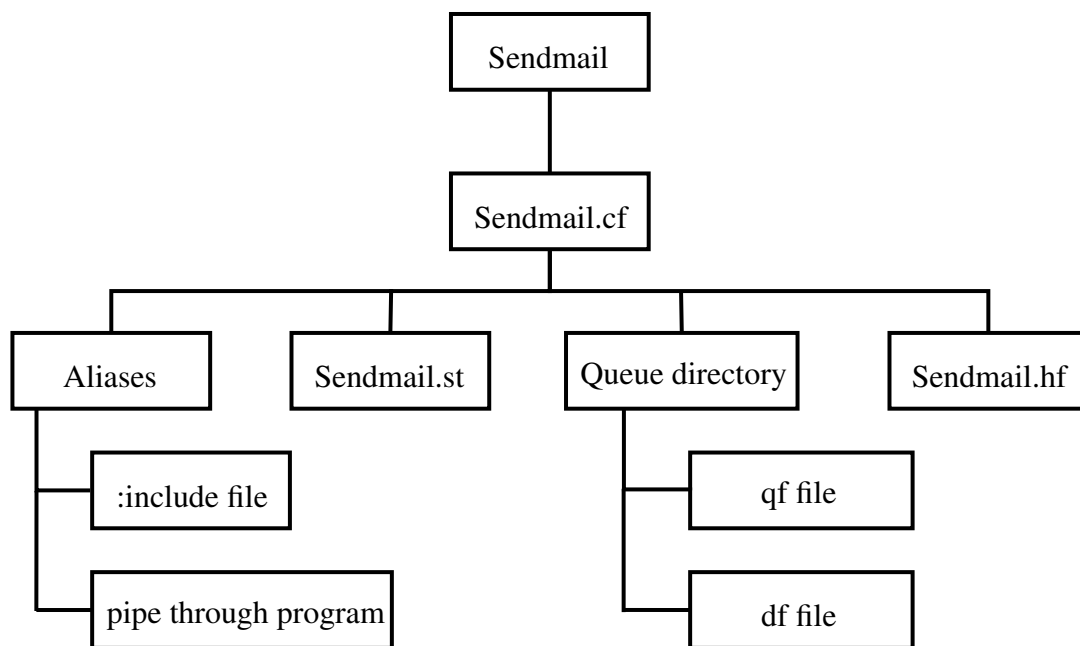


Figura 3.5: As Funcionalidades são Definidas no Arquivo Sendmail.cf.

Fonte: Costales e Allman [58]

Os arquivos ou pastas definidos no *sendmail.cf* são em geral descritos com o caminho completo, como por exemplo, o arquivo *aliases*. A letra “O” no início da linha indica que trata-se de uma opção de configuração.

```
O AliasFile=/etc/mail/aliases
```

O arquivo *aliases* contém configurações do processo de *Aliasing*. Este processo converte o nome de um endereço em outro nome. Por exemplo, nomes genéricos como *sysadmin* ou *abuse* podem ser convertidos em nomes reais [60].

```
# Basic system aliases -- these MUST be present.
MAILER-DAEMON: postmaster
postmaster: root

# General redirections for pseudo accounts.
bin: root
daemon: root
games: root
ingres: root
nobody: root
```

```
system:      root
toor:        root
uucp:        root

# Well-known aliases.
manager:     root
dumper:      root
webmaster:   alice
abuse:       bob
sysadmin:    bob
diretoria:   alice,bob,trend,john

#old lists
oldguys: :include: /usr/local/mail/oldguys

#help
ftphelp: |/usr/local/bin/sendhelp
```

Figura 3.6: Exemplo de um Arquivo *Aliases*.

A Figura 3.6 mostra uma impressão de tela de um arquivo *aliases* padrão da distribuição *Linux Slackware* [61] com pequenas alterações. O nome à esquerda de cada linha será alterado pelo nome que se encontra no lado direito. Observam-se entradas nas quais um nome será substituído por outro, ou por um grupo de nomes, ou pelo conteúdo de um arquivo; e um nome cujas mensagens enviadas a ele serão redirecionadas a um aplicativo.

O arquivo *sendmail.st* armazena informações sobre as quantidades e tamanhos das mensagens recebidas e enviadas pelo MTA. Estas informações são exibidas por meio do comando *mailstats*. O arquivo *sendmail.hf* contém informações de ajuda para comandos do protocolo SMTP.

O diretório da fila de mensagens, também chamado de *queue*, é o local onde as mensagens que não puderam ser entregues ficam armazenadas temporariamente. Periodicamente, o *Sendmail* tenta entregar as mensagens até um limite definido pelo administrador. Este tempo varia entre dois a quatro dias. O intervalo de tempo entre as tentativas também é definido pelo administrador, sendo comum o valor de quinze minutos [60].

A opção `QueueDirectory` dentro do arquivo *sendmail.cf* define o local de armazenamento temporário:

```
O QueueDirectory=/var/spool/mqueue
```

Quando uma mensagem é enfileirada, ela é dividida em duas partes. Cada parte é salva em um arquivo diferente. O cabeçalho da mensagem é salvo em arquivos que começam com as letras `qf`, enquanto que o corpo da mensagem é salvo em arquivos cujos nomes começam com as letras `df` [60].

Outra função do *Sendmail* é entregar as mensagens para os usuários locais. A entrega da mensagem é feita acrescentando a mensagem no final do arquivo que armazena a caixa postal do usuário. Esta entrega não é feita normalmente pelo *Sendmail* e, sim, por outro programa que recebe a tarefa do MTA.

Encontra-se no arquivo *sendmai.cf* dois agentes de entrega responsáveis por esta tarefa

```
Mlocal,      P=/usr/bin/procmail, F=lsDFMAw5:/|@qSPfhn9, S=EnvFromL/HdrFromL, R=EnvToL/HdrToL,
Mprog,      P=/bin/sh, F=lsDFMoqeu9, S=EnvFromL/HdrFromL, R=EnvToL/HdrToL, D=$z:/,
```

Enquanto o `/usr/bin/procmail` é usado para adicionar a mensagem na caixa postal do usuário, o `/bin/sh` aciona outros programas para fazer a entrega.

Quando o *Sendmail* determina que o endereço do destinatário não é local, a mensagem deve ser encaminhada para outro servidor. A seguinte linha define como as mensagens serão enviadas para outras máquinas:

```
Msmtp,      P=[IPC], F=mDFMuX, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP, E=\r\n, L=990,
```

O valor `[IPC]` é equivalente ao valor `[TCP]`. O *Sendmail*, ao usar uma rede TCP/IP para enviar mensagens, envia primeiro o remetente contido no envelope para o outro servidor. Uma vez que o endereço do remetente foi aceito, o *Sendmail* local envia a lista dos destinatários. O servidor do destinatário aceita ou rejeita cada endereço. Caso algum endereço de destinatário seja aceito, o cabeçalho e o corpo da mensagem são enviados juntos [60].

3.2.2 Configuração do *Sendmail*

A localização do arquivo de configuração do *Sendmail* varia de acordo com o sistema operacional. Na distribuição Linux Slackware 12.1 [61] encontra-se em `/etc/mail/`. O conteúdo deste arquivo possui uma aparência ilegível para a maioria das pessoas, tornando difícil a sua manutenção.

Com o uso do pré-processador *m4*, a criação do arquivo `sendmail.cf` ficou mais simples. O *m4* é um programa que substitui textos, principalmente em programação e configuração de aplicações [62]. O programa *m4* produz um arquivo de configuração do *Sendmail* por meio

do processamento de um arquivo com extensão `.mc` (de *macro configuration*). O programa *m4* coleta as definições dos *macros* no arquivo de entrada e substitui os valores a elas associados, produzindo uma saída no formato `.cf`.

O uso dos macros nos arquivos tipo `.mc` tem o seguinte formato:

```
define(macroname, value)
```

sendo *macroname* o valor de nome simbólico do *macro*. O conteúdo de *value* pode ser qualquer texto. Todo arquivo do tipo `.mc` requer um mínimo de informação. A Tabela 3.1 mostra dois itens necessários e dois itens recomendados para um arquivo de configuração de *macro*.

Tabela 3.1: Itens *m4* Necessários e Recomendados

Item		Descrição
OSTYPE()	Necessário	Suporte para o sistema operacional
MAILER()	Necessário	Agentes de entrega
DOMAIN()	Recomendado	Informações comuns do domínio
FEATURE()	Recomendado	Soluções para necessidades especiais

Fonte: Costales e Allman [58]

Sistemas operacionais como AIX, BSD, Solaris, Linux entre outros, têm suporte com o comando `OSTYPE()`. Todo arquivo `.mc` deve declarar o sistema operacional com este comando. Na pasta `cf` do pacote de distribuição do *Sendmail* existe uma pasta `ostype` que na distribuição Slackware 12.1, fica em `/usr/share/sendmail/cf/ostype`, oferecendo suporte aos sistemas operacionais exibidos na Figura 3.7:

a-ux.m4	darwin.m4	hpux9.m4	osf1.m4	solaris8.m4
aix3.m4	dgux.m4	irix4.m4	powerux.m4	sunos3.5.m4
aix4.m4	domainos.m4	irix5.m4	ptx2.m4	sunos4.1.m4
aix5.m4	dragonfly.m4	irix6.m4	qnx.m4	svr4.m4
altos.m4	dynix3.2.m4	isc4.1.m4	riscos4.5.m4	ultrix4.m4
amdahl-uts.m4	freebsd4.m4	linux.m4	sco-uw-2.1.m4	unicos.m4
bsd4.3.m4	freebsd5.m4	maxion.m4	sco3.2.m4	unicosmk.m4
bsd4.4.m4	freebsd6.m4	mklinux.m4	sinix.m4	unicosmp.m4
bsdi.m4	gnu.m4	mpeix.m4	solaris2.m4	unixware7.m4
bsdi1.0.m4	hpux10.m4	nextstep.m4	solaris2.ml.m4	unknown.m4
bsdi2.0.m4	hpux11.m4	openbsd.m4	solaris2.pre5.m4	uxpds.m4

Figura 3.7: Lista de Sistema Operacionais Suportados pelo *m4*

Os agentes de entrega não são declarados automaticamente, sendo necessário uma especificação. Para especificar um agente é necessário usar o macro:

```
MAILER( 'local' )
```

Este macro permite suporte para os agentes de entrega `local` e `prog`. Esta é uma configuração mínima e necessária. A Figura 3.8 ilustra os agentes de entrega disponíveis na distribuição Linux Slackware 12.1.

```
cyrus.m4      fax.m4      mail11.m4    pop.m4      qpage.m4    usenet.m4
cyrusv2.m4    local.m4    phquery.m4  procmail.m4 smtp.m4      uucp.m4
```

Figura 3.8: Lista de Agentes de Entrega

O *Sendmail* oferece ainda várias funcionalidades. Para incluir uma funcionalidade à configuração, utiliza-se o macro:

```
FEATURE(keyword)
```

```
FEATURE(keyword, argument)
```

Estas declarações dentro do arquivo `.mc` instruem o `m4` ler o arquivo `/usr/share/sendmail/cf/feature/keyword.m4` no local do comando e produzir as linhas de configurações associadas à funcionalidade. A Figura 3.9 ilustra as funcionalidades presentes na distribuição Slackware 12.1.

<code>accept_unqualified_senders.m4</code>	<code>greet_pause.m4</code>	<code>preserve_luser_host.m4</code>
<code>accept_unresolvable_domains.m4</code>	<code>ldap_routing.m4</code>	<code>promiscuous_relay.m4</code>
<code>access_db.m4</code>	<code>limited_masquerade.m4</code>	<code>queuegroup.m4</code>
<code>allmasquerade.m4</code>	<code>local_lmtp.m4</code>	<code>ratecontrol.m4</code>
<code>always_add_domain.m4</code>	<code>local_no_masquerade.m4</code>	<code>redirect.m4</code>
<code>authinfo.m4</code>	<code>local_procmail.m4</code>	<code>relay_based_on_MX.m4</code>
<code>badmx.m4</code>	<code>lookupdotdomain.m4</code>	<code>relay_entire_domain.m4</code>
<code>bestmx_is_local.m4</code>	<code>loose_relay_check.m4</code>	<code>relay_hosts_only.m4</code>
<code>bitdomain.m4</code>	<code>mailertable.m4</code>	<code>relay_local_from.m4</code>
<code>blacklist_recipients.m4</code>	<code>masquerade_entire_domain.m4</code>	<code>relay_mail_from.m4</code>
<code>block_bad_helo.m4</code>	<code>masquerade_envelope.m4</code>	<code>require_rdns.m4</code>
<code>compat_check.m4</code>	<code>msh.m4</code>	<code>smrsh.m4</code>
<code>conncontrol.m4</code>	<code>mtamark.m4</code>	<code>stickyhost.m4</code>
<code>delay_checks.m4</code>	<code>no_default_msa.m4</code>	<code>use_client_ptr.m4</code>
<code>dnsbl.m4</code>	<code>nocanonify.m4</code>	<code>use_ct_file.m4</code>
<code>domaintable.m4</code>	<code>notsticky.m4</code>	<code>use_cw_file.m4</code>
<code>enhdnsbl.m4</code>	<code>nouucp.m4</code>	<code>uucpdomain.m4</code>
<code>generics_entire_domain.m4</code>	<code>nullclient.m4</code>	<code>virtuser_entire_domain.m4</code>
<code>genericstable.m4</code>	<code>preserve_local_plus_detail.m4</code>	<code>virtusertable.m4</code>

Figura 3.9: Lista de Funcionalidades Disponíveis

3.3 Filtrando Mensagens com o MILTER

O *Sendmail Content Management API* (*Application Programming Interface*) é também chamado de **MILTER**, que é uma composição das palavras *mail* e *filter*. O **MILTER** está presente no *Sendmail* desde a versão 8.11.x como inclusão experimental e foi lançado oficialmente na versão 8.12.0 [59]. Até a versão 8.13 o *Sendmail* tem que ser compilado com o parâmetro `-DMILTER` para habilitar o suporte ao **MILTER** [60].

Os principais objetivos da API do **MILTER** são: segurança, confiabilidade, simplicidade e desempenho [59].

A **segurança** é melhorada pela não necessidade de privilégios de superusuário (*root*) para executar o **MILTER**. Isto simplifica a programação e diminui o impacto de brechas de segurança do programa de filtro.

Para manter a **confiabilidade** da entrega das mensagens, o congelamento ou término de execução do programa não interrompem a entrega das mensagens. Perante uma falha de filtro, o *Sendmail* pode agir como se o filtro não estivesse presente, retornar um erro 4xx do protocolo SMTP ou enfileirar a mensagem para processá-la mais tarde.

A API do **MILTER** procura a **simplicidade** para que o desenvolvimento de novos filtros não seja mais difícil do que o absolutamente necessário. E os filtros não devem impactar no **desempenho** geral do MTA.

3.3.1 Configurando o MILTER no Sendmail

Para configurar no *Sendmail* um **MILTER** (*Mail Filter*), deve-se definir o parâmetro $S =$ que informa ao MTA como conectar com o **MILTER**. A conexão pode ser feita com Unix *sockets* ou IP *sockets* [59].

O parâmetro opcional $F =$ define o comportamento do *Sendmail* quando o **MILTER** estiver indisponível. A Tabela 3.2 mostra os possíveis valores para $F =$ e o respectivo significado.

Tabela 3.2: Comportamento do *Sendmail* na Ausência do **MILTER**

$F =$	Significado
R	Rejeitar a conexão se o MILTER não estiver disponível
T	Falha temporária se o filtro não estiver disponível

No Apêndice A.1 tem-se o arquivo `mc` usado neste trabalho para criar o arquivo de configuração do *Sendmail*, com os macros para o uso de um *MILTER*. Para gerar o arquivo `sendmail.cf` a partir do arquivo `.mc` deve-se utilizar o seguinte comando:

```
m4 < arquivo.mc >sendmail.cf
```

3.3.2 Políticas de Filtragem

O *MILTER* permite que programas de terceiros acessem as mensagens eletrônicas durante seu processamento pelo MTA. Desta maneira o *MILTER* pode analisar e até alterar o conteúdo das mensagens [59].

Ele pode, ainda, analisar endereços do protocolo de rede IP, elementos do envelope SMTP, elementos do cabeçalho e do corpo da mensagem. Com o *MILTER* as políticas de filtragem podem ficar centralizadas no arquivo de configuração do MTA, onde definem-se quais e em que ordem os filtros devem ser aplicados.

Ao administrador de sistema compete configurar o MTA para inicialmente verificar a mensagem por meio de um mecanismo antivírus, usar um serviço de detecção de *spam* e, finalmente, anexar a mensagem de rodapé nas mensagens que atenderem a certo critério.

É importante notar que a política de filtro é definida pelo administrador do MTA e não pelo autor do filtro. Cabe ao autor recomendar uma política adequada para um uso eficiente do *MILTER*. A política inclui definir a ordem com que os filtros são aplicados, como o *MILTER* comunica com o MTA e que ação deve ser tomada pelo MTA caso o filtro fique indisponível.

3.3.3 Comunicação do Filtro com o MTA

Os *MILTERs* rodam como processos separados do *Sendmail*, e como consequência, não necessitam de permissão de superusuário para serem executados. Isso evita potenciais problemas de segurança. Outra consequência é que eventuais falhas de um *MILTER* não afeta o funcionamento de outro *MILTER*.

A Figura 3.10 é uma esquematização de uma rede com um conjunto de filtros, seu MTA e canais de comunicação com outros MTAs. Cada *MILTER* comunica-se com vários MTAs, localmente ou remotamente, usando várias linhas de execução.

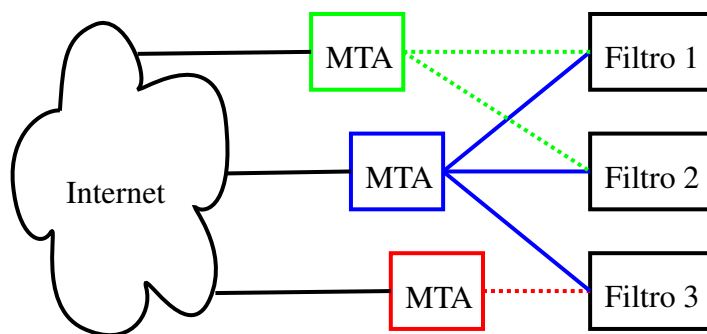


Figura 3.10: Um Conjunto de MTA Interagindo com um Conjunto de *MILTERs*

Na Figura 3.11 pode-se visualizar um *MILTER* processando mensagens de dois MTAs. A biblioteca *libmilter* implementa o protocolo de comunicação. Ela aceita a conexão de diversos MTAs, transfere informações relevantes para o *MILTER* por meio de *callbacks* e, de acordo com o código de retorno, o *MILTER* produz respostas adequadas para o MTA.

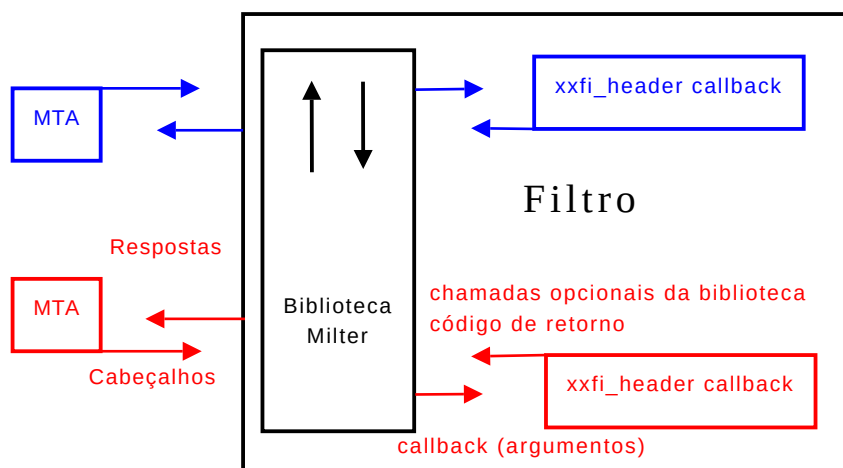


Figura 3.11: Requisições Simultâneas de Dois MTAs

O Algoritmo 3.12 descreve o processo de filtragem de N conexões de diversos MTAs [59]. Os *callbacks* são mostrados ao lado do estágio do processo em que eles são chamados. Apesar de não estar definido, o processamento pode ser interrompido a qualquer instante, devolvendo o controle para MESSAGE, sendo necessária a chamada do *callback* `xxfi_abort`.

```

For each of N connections
{
    For each filter
        process connection/helo (xxfi_connect, smfi_helo)
MESSAGE: For each message in this connection (sequentially)
    {
        For each filter
            process sender (xxfi_envfrom)
        For each recipient
        {
            For each filter
                process recipient (xxfi_envrcpt)
            }
        For each filter
        {
            process DATA (xxfi_data)
            For each header
                process header (xxfi_header)
            process end of headers (xxfi_eoh)
            For each body block
            – process this body block (xxfi_body)
            process end of message (xxfi_eom)
        }
    }
    For each filter
        process end of connection (xxfi_close)
}

```

Figura 3.12: Algoritmo de Operação do MILTER

3.4 Autenticação de Remetentes

Nesta seção são resumidos os aspectos relevantes dos trabalhos relacionados à autenticação de remetentes. A preocupação com a autenticidade da identidade dos remetentes cresce e novas propostas contra os endereços forjados são criadas.

O PGP [63] idealizado por Phil Zimmerman tem por objetivo fornecer confidencialidade e autenticidade das mensagens eletrônicas [18]. Essencialmente, Zimmerman selecionou os melhores algoritmos disponíveis e os integrou estes algoritmos em uma aplicação de uso geral, in-

dependente de plataforma e processador. Além de algoritmos consagrados, o PGP fundamenta-se em uma rede de confiança na qual um usuário compartilha chaves públicas confiáveis de sua base de dados com outros usuários [18].

Em relação ao PGP, o OpenPGP acrescenta a possibilidade da presença de CA cujas assinaturas podem certificar a autenticidade de outras assinaturas. Funciona nos agentes usuários, independente dos MTAs, ou seja, é um sistema fim-a-fim. Apesar da eficiência em detectar mensagens forjadas, tem como fraquezas um complexo gerenciamento de chaves públicas e o desperdício de tempo e recursos de rede com o transporte de mensagens até o usuário. A Figura 3.13 ilustra o funcionamento da autenticação via assinatura digital do OpenPGP.

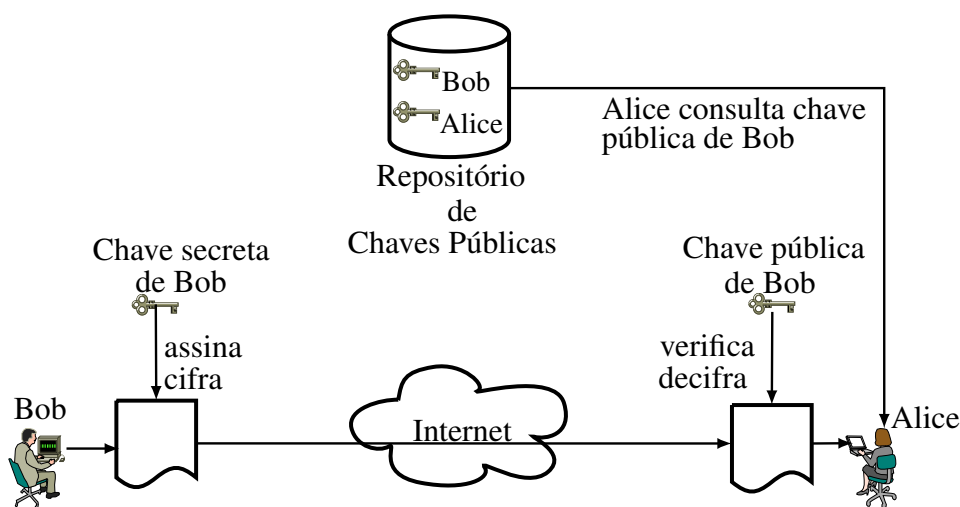


Figura 3.13: Funcionamento da Assinatura Digital do OpenPGP

Outro método pesquisado, o SPF (*Sender Policy Framework*) [64], tem uma abordagem oposta ao PGP, posto que ele transfere aos servidores a tarefa de verificar a autenticidade da mensagem. Na verdade, o SPF não verifica a autenticidade do remetente e, sim, se o cliente SMTP pertence a uma lista de hospedeiros autorizados.

A Figura 3.14 esquematiza o funcionamento do SPF. Ao receber uma mensagem, o destinatário, com base nas informações provenientes do envelope (Mail From, Ehlo e Helo), extrai o endereço IP do Cliente SMTP e o domínio do remetente.

Os administradores do domínio publicam previamente, por meio dos registros DNS, a relação de hospedeiros autorizados a enviar mensagens do domínio. O Servidor SMTP, de posse das informações do envelope da mensagem e dos dados publicados pelos registros DNS, decide se a mensagem veio de um remetente legítimo ou não.

Uma grande vantagem deste *framework* é sua simplicidade e facilidade de implantação. Um domínio dificulta a criação de mensagens forjadas, simplesmente publicando uma lista de

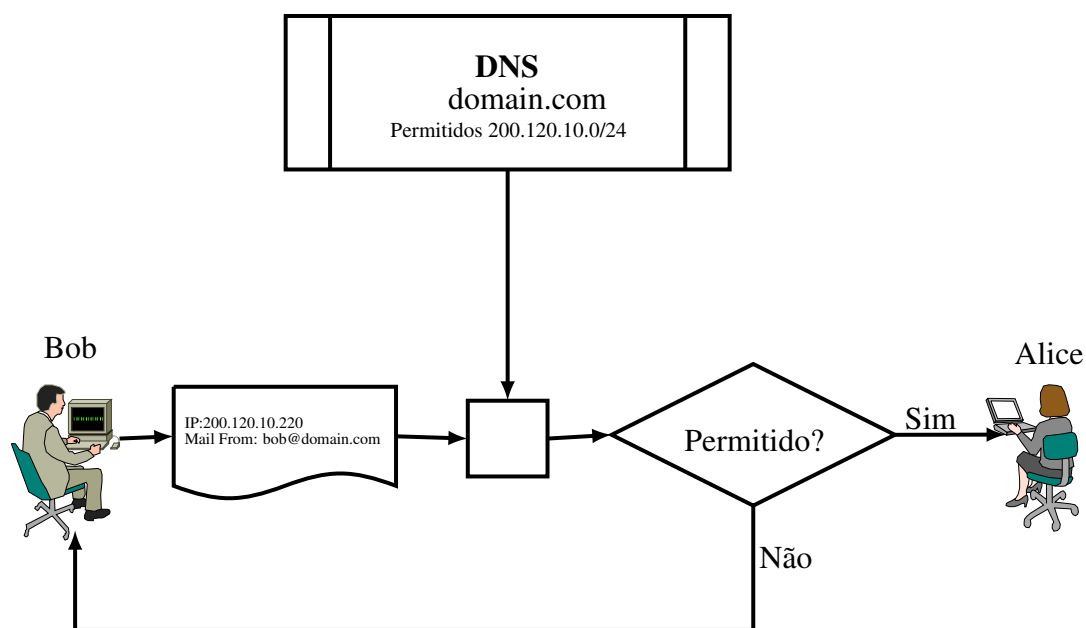


Figura 3.14: Esquema de Funcionamento do SPF

hospedeiros autorizados a serem clientes SMTP de mensagens do domínio. A verificação da autenticidade das mensagens e posterior ação são de responsabilidade do servidor SMTP do destinatário.

O SPF não admite o repasse de mensagens, podendo rejeitar mensagens válidas. Pode também autorizar uma mensagem forjada, desde que o adversário use um hospedeiro que esteja na lista de permitidos. O SPF ainda obriga os usuários a utilizarem somente os clientes SMTP que constem nas listas dos permitidos, o que pode, da mesma forma que o SPF, ser inconveniente para usuários em trânsito.

O autor do *Sendmail* Eric Allman, juntamente com pesquisadores da *Cisco Systems*, *Yahoo*, *PGP Corporation* e outros, em um trabalho conjunto para o IETF propôs o DKIM (*Domain Keys Internet Mail*) [65].

O DKIM propõe o uso de mecanismos criptográficos para evitar mensagens forjadas, assinando digitalmente as mensagens. Diferentemente do SPF, que utiliza o envelope, o DKIM assina informações contidas no cabeçalho da mensagem. A Figura 3.15 ilustra o funcionamento do DKIM [65].

O DKIM assina os campos *From*, *To*, *Subject* e outros campos juntamente com um *Hash* da mensagem. As chaves públicas associadas ao domínio são publicadas pelos registros DNS, por meio de seletores que permitem flexibilidade de múltiplas chaves por domínio. O DKIM obriga os usuários a utilizar os clientes SMTP do domínio, o que pode ser inconveniente em caso de usuários em trânsito.

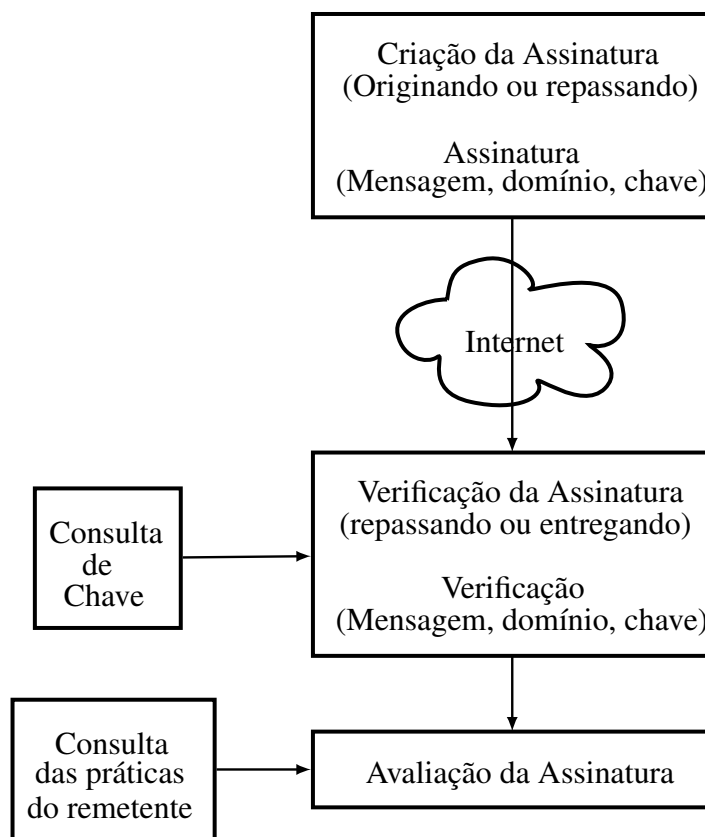


Figura 3.15: Esquema de Funcionamento do DKIM

Fonte: Allman et al. [65]

Por assinar campos do cabeçalho, o DKIM não tem problemas com o repasse de mensagens, porém alterações no campo `Subject` e no corpo da mensagem, comuns em lista de discussão, podem invalidar a assinatura [66].

A assinatura da mensagem é feita pelo servidor SMTP, com uma única chave secreta, o que permite que usuários deste servidor forjem a identidade de outros usuários. O DKIM atribui ao domínio a responsabilidade sobre os usuários e somente reclamações ou inspeções profiláticas permitiriam medidas restritivas ou punitivas aos infratores. Um mecanismo de autenticação baseada em remetente tornaria mais difícil a implementação, pois seria necessário adaptações dos MTAs.

Com uma abordagem semelhante ao PGP, o LES [20] propõe um mecanismo de assinatura de mensagens utilizando IBS. Assim como o PGP, o LES baseia-se em um esquema fim-a-fim, de acordo com o qual os usuários finais são responsáveis pela assinatura e verificação, porém o LES propõe somente o uso de assinaturas digitais sem a cifragem da mensagem.

O uso de IBS simplifica o gerenciamento de chaves em relação ao DKIM e não permite que um usuário do mesmo domínio falsifique a assinatura de outro usuário. Os parâmetros

públicos do domínio são divulgados por meio dos registros DNS, assim como o SPF e DKIM.

Tanto o DKIM como o LES são flexíveis quanto ao algoritmo criptográfico para assinatura e ambos adotam o RSA. Com o uso de dispositivos móveis com baixo poder de processamento e a necessidade crescente de chaves maiores para garantir segurança, o uso do RSA pode em breve tornar-se um problema.

Capítulo 4

Aspectos de Implementação e Resultados Obtidos

Neste capítulo é apresentado a proposta de um esquema de autenticação de remetente em servidor de e-mail, com o uso de assinatura digital baseada em identidade, tendo em vista o objetivo principal deste trabalho que é propiciar aos MTAs a capacidade de autenticar o endereço do remetente [67].

Esta proposta teve em vista possibilitar que o servidor identifique com facilidade a chave pública do remetente, com baixo impacto no desempenho do MTA ao verificar a assinatura. Além disso, o esquema proposto deve funcionar com a infra-estrutura de *e-mail* existente, isto é, com integração de novas funcionalidades sem alterar essa aplicação.

Nos esquemas de autenticação fim-a-fim, como o OpenPGP e LES, a mensagem fica armazenada no servidor SMTP até o destinatário acessá-la. Ao ser acessada a mensagem é transmitida até o hospedeiro do destinatário. A assinatura será verificada pelo usuário após ela ser recebida do transmissor.

Neste esquema, que está sendo apresentado, a verificação da assinatura da mensagem é feita pelo servidor SMTP. A escolha do servidor como ponto para verificar a autenticidade do remetente justifica-se não só pelas ameaças diretas, como *phishing* [11], que uma mensagem pode oferecer ao destinatário, mas também pelo desperdício de recursos de armazenamento e de rede.

Como conseqüências destes desperdícios, pode-se citar por exemplo, a perda de mensagens. Em muitos sistemas há limite de espaço destinado ao armazenamento das mensagens nas caixas postais. Devido ao alto volume de SPAM, existe o risco do espaço de armazenamento

esgotar-se e mensagens legítimas serem recusadas pela falta de espaço nas caixas postais.

Outra consequência de desperdícios acomete usuários de recursos de redes escassos e onerosos para acessar o correio eletrônico, como por exemplo os dispositivos móveis, no caso dos celulares que possuem conexões limitadas, muitas vezes cobradas por volume de dados transportados. Assim, para esses clientes é inconveniente receber a mensagem para então verificar-se sua legitimidade.

Para evitar estes desperdícios, alguns mecanismos como SPF e DKIM trabalham com a verificação na esfera do servidor. O SPF verifica simplesmente se o endereço IP do cliente SMTP tem origem em redes com permissão de envio. De maneira semelhante, o DKIM verifica a permissão por meio de assinatura digital.

Estes mecanismos ocasionam problemas para usuários móveis que, por algum motivo, não podem acessar os servidores com permissão de envio de mensagens. Também, por trabalhar somente no âmbito de servidores, remetentes podem ainda forjar o endereço de outros, desde que estejam na rede de endereços permitidos.

A existência destes problemas indica que os servidores devem verificar e filtrar mensagens forjadas e os remetentes devem assinar as mensagens enviadas. Tem-se, portanto, a assinatura na esfera do usuário e a verificação na dos servidores.

O IBS proposto inicialmente por Shamir [46] traz propriedades atrativas em relação aos mecanismos tradicionais de assinatura digital: é simples para o servidor descobrir a chave pública do usuário e não há necessidade de um certificado, pois a certificação está implícita na chave pública.

A verificação das assinaturas digitais dos remetentes pelos servidores cria para estes uma dificuldade extra, visto que os servidores devem encontrar a chave pública do remetente. Entretanto, essa informação pode ser buscada em um repositório de chaves públicas, ou o próprio remetente informa qual a sua chave pública. A primeira alternativa insere uma carga indesejada de trabalho para o servidor, o que a torna impraticável. A segunda alternativa, por sua vez, necessita de uma terceira parte confiável para confirmar que a chave pública fornecida realmente está associada ao remetente da mensagem, e necessita, ainda, verificar se a chave fornecida não está em uma lista de certificados revogados.

A Figura 4.1 ilustra um esquema de autenticação de remetente. O usuário final, por meio do MUA, assina sua mensagem digitalmente, de qualquer ponto da *Internet*, e os servidores SMTP verificam e filtram as mensagens antes de entregá-las para os destinatários.

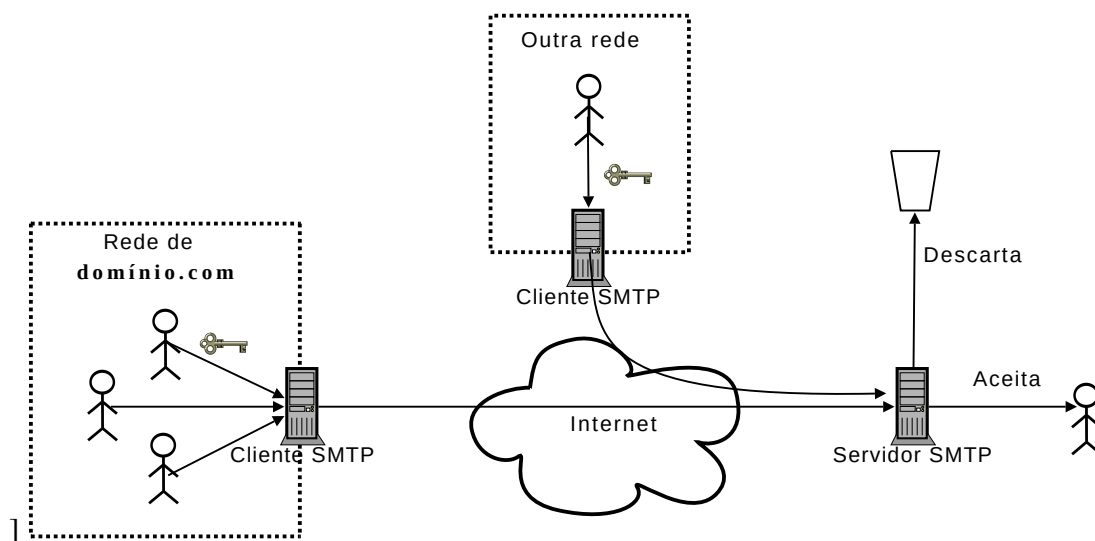


Figura 4.1: Esquema de Funcionamento de Filtro de Remetente

Apresenta-se a seguir, o funcionamento do esquema proposto, com detalhamento da sequência do processamento das mensagens.

4.1 Operação do Esquema Proposto

Inicialmente, usuário final requisita serviços de correio eletrônico que são compostos pelo MUA e pelo MTA. O MUA interage diretamente com o usuário final e os MTAs interagem entre si e com o MUA. Por sua vez, os componentes serviços de correio eletrônico utilizam-se da assinatura digital para identificar as mensagens de maneira singular e verificar posteriormente esta identificação.

Para assinar e verificar as mensagens, os serviços de assinatura digital necessitam gerar e armazenar parâmetros públicos e extrair chaves secretas. O resultado da verificação é remetido para o MTA que a repassa para o filtro analisarem a informação, que é comparada com informações de bases de reputação de remetentes, para permitir ou negar a entrega da mensagem.

O funcionamento do esquema é ilustrado na Figura 4.2.

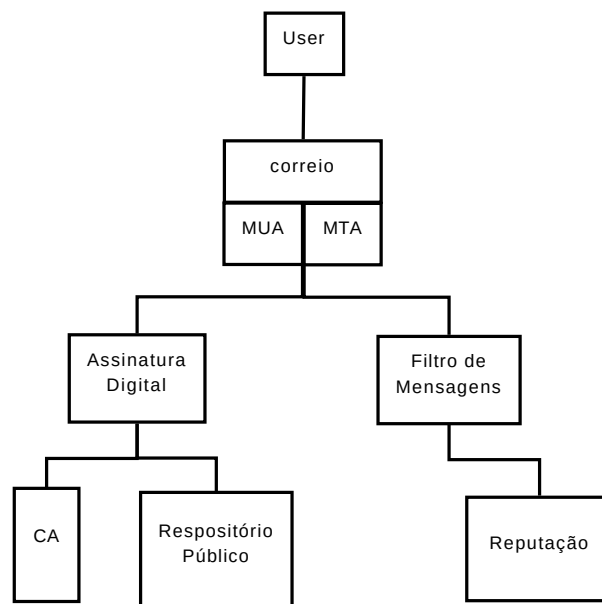


Figura 4.2: Esboço de Operação

O usuário, por meio do MUA, requisita serviços de correio. Nesta fase, após produzir a mensagem, o MUA tem a tarefa de assinar a mensagem. Para tal, o MUA solicita ao bloco de assinatura digital uma chave secreta, informando qual será a chave pública temporária associada à assinatura, conforme ilustrado na Figura 4.3.

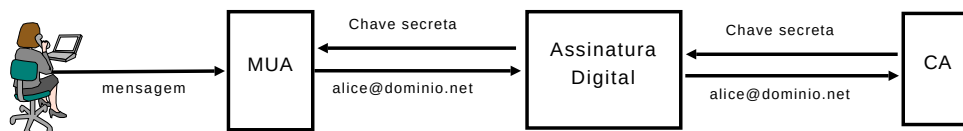


Figura 4.3: MUA Requisita ao CA Chave Secreta para Determinada Chave Pública

O responsável pela extração da chave secreta relacionada com a chave pública fornecida pelo MUA é a CA. Utilizando um parâmetro secreto do domínio do endereço do usuário e a chave pública inicial, o CA faz um cálculo e retorna o resultado deste cálculo como a chave secreta, que é, então, entregue ao MUA. É papel do CA manter atualizados os dados armazenados no repositório público.

O MUA, então de posse da chave secreta, assina a mensagem e solicita ao MTA que transmita a mensagem ao destinatário. O MTA do destinatário, ao receber a mensagem, necessita verificar a sua assinatura. Para tanto, determina a chave pública a partir da mensagem e solicita ao bloco de assinatura digital o parâmetro público associado ao domínio do remetente. O repositório público, ao receber a solicitação, retorna os parâmetros previamente divulgados pelo CA.

De posse da chave pública, dos parâmetros públicos, da mensagem e assinatura, o MTA do destinatário verifica a assinatura e o resultado da verificação é encaminhado para o filtro de mensagens, conforme ilustrado na Figura 4.4.

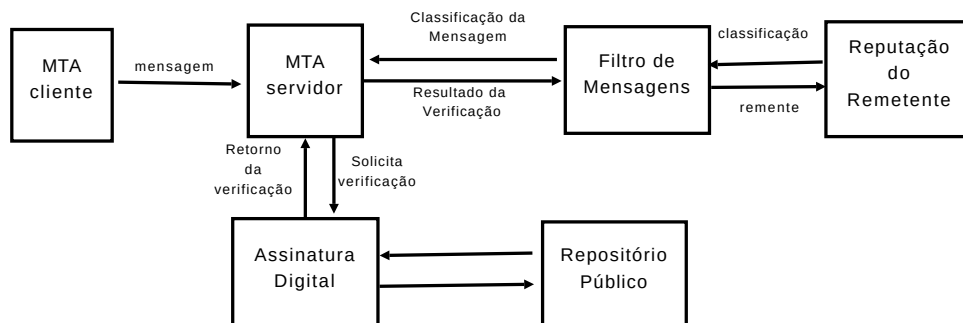


Figura 4.4: MTA Requisita ao Filtro a Análise da Mensagem

O filtro de mensagens, uma vez que o endereço do remetente não é forjado, pode classificar a mensagem de acordo com a reputação do remetente. Esta classificação é devolvida ao MTA para que ele possa aceitar, rejeitar ou simplesmente marcar a mensagem. A Figura 4.5 ilustra o fluxo do processamento de uma mensagem.

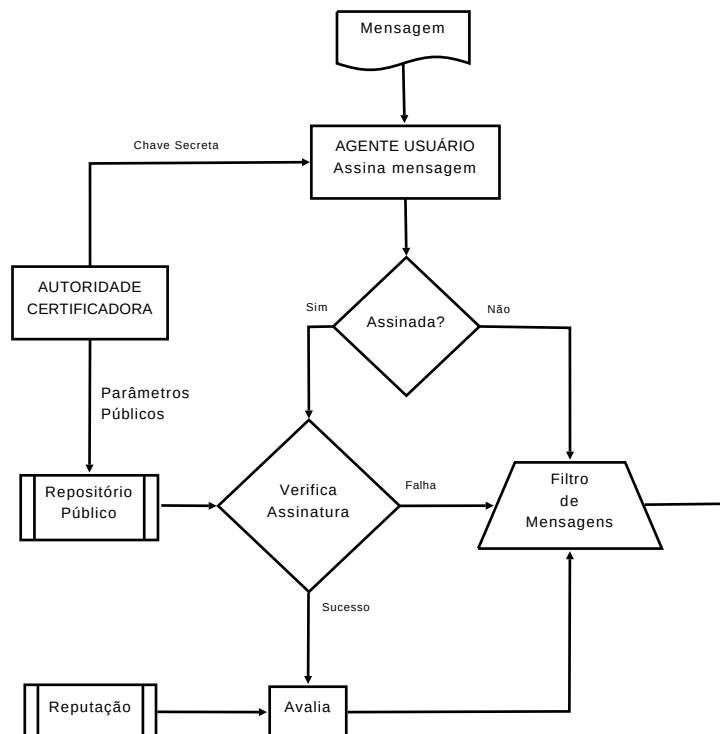


Figura 4.5: Processamento da Mensagem

Designar o servidor como responsável pela verificação da autenticidade dos remetentes acarreta alguns problemas que devem ser resolvidos, tais como o mecanismo de verificação

deve ter um impacto pequeno no desempenho do MTA e o MTA deve ser capaz de determinar com facilidade a chave pública do remetente e aproveitar a infra-estrutura existente.

Para facilitar a determinação da chave pública pelo servidor, escolheu-se o IBS, pois assinaturas baseadas em identidade permitem ao servidor SMTP determinar com facilidade a chave pública e relacioná-la com o seu proprietário. Esta característica facilita ao servidor determinar a autenticidade do remetente no âmbito do usuário.

Além da facilidade de determinar chaves públicas, o IBS pode utilizar ECC. A vantagem do ECC em relação ao RSA diz respeito à eficiência computacional. Nos cálculos de assinatura e verificação, não há diferença significativa em termos de esforço computacional [42], porém para atingirem patamares semelhantes de segurança, os mecanismos devem utilizar diferentes tamanhos de chaves. Este fato se dá em função da diferença de eficiência entre algoritmos utilizados para resolução do problema matemático dos sistemas criptográficos.

Para garantir um patamar de segurança adequado, deve-se optar por chaves com tamanho que torne computacionalmente impraticável a resolução do problema matemático associado ao esquema criptográfico. Assim, a escolha de esquemas IBS baseados em emparelhamentos sobre curvas elípticas foi feita para que a crescente necessidade de chaves criptográficas maiores não cause impacto no desempenho do MTA.

Independentemente do mecanismo criptográfico escolhido para assinatura das mensagens, é necessária a presença de um CA como terceira parte confiável. É necessária ainda a escolha de um mecanismo para publicar os parâmetros públicos das entidades confiáveis responsáveis pela geração das chaves secretas dos usuários. Pela simplicidade de uso e para não ser preciso criar uma infra-estrutura de PKG (*Private Key Generator*), foi escolhido o DNS. Apesar do DNS possuir vulnerabilidades conhecidas [68], sua simplicidade e escalabilidade o tornam atrativo. A proposta de DNS seguro é uma alternativa a ser usada [69].

A junção dos elementos construtivos até agora expostos permite que se apresente um esquema de autenticação de remetente com assinaturas baseadas em identidade, conforme ilustrado na Figura 4.6.

Em (1), o servidor de chaves disponibiliza parâmetros do `domínio.com`; em (2) Alice requisita sua chave secreta para uma data específica; em (3) Alice assina a mensagem com a sua chave secreta; em (4) o servidor SMTP identifica chave pública de Alice a partir dos cabeçalhos da mensagem; em (5) o servidor SMTP requisita os parâmetros do `domínio.com` e em (6) o servidor verifica a assinatura da mensagem de Alice.

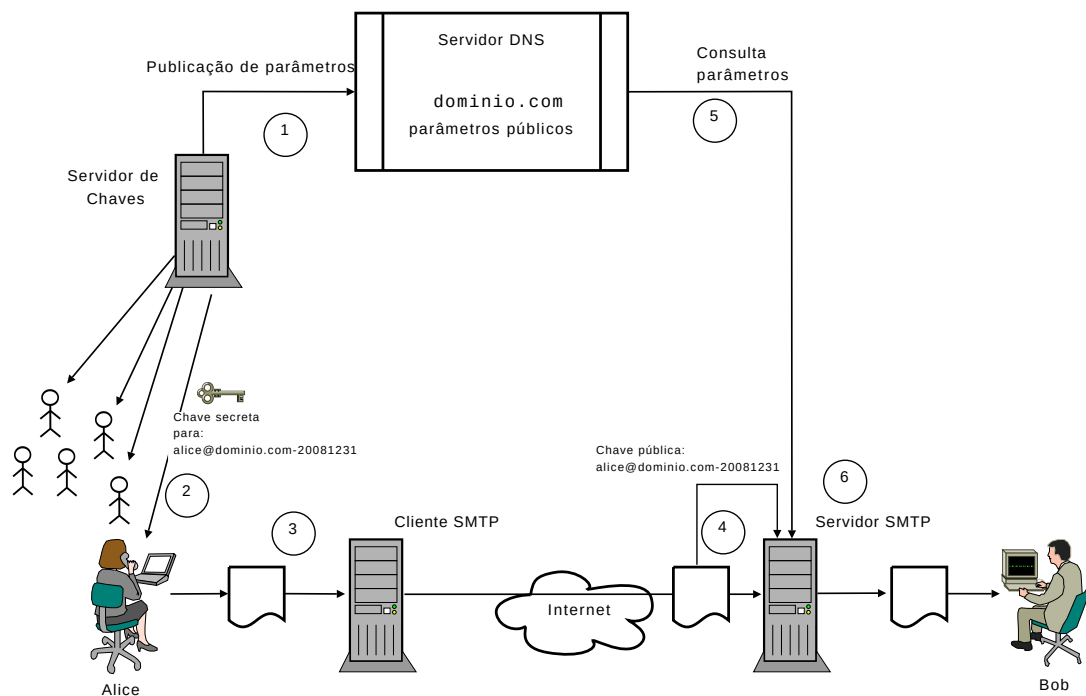


Figura 4.6: Esquema de Autenticação de Remetente

4.2 Ambiente de Desenvolvimento

Os computadores utilizados no ambiente de desenvolvimento e na realização dos testes, bem como o utilizado como servidor SMTP, possuem as seguintes características de hardware e software:

- Dell Latitude D530;
- Processador T7250 Core 2 Duo de 2GHz;
- Cache L1 de 2MB;
- Memória RAM de 2GB 667MHz;
- Sistema operacional Linux kernel 2.6.24-5-smp [70];
- Distribuição Slackware 12.1 [61];
- MTA utilizado foi o sendmail 8.14.2 [59];
- Aplicativo DNS - BIND (*Berkeley Internet Name Domain*)(citar) versão 9.4.2 [];
- Compilador GCC (*Gnu Compiler Collection*) versão 4.2.3 [71];

- Biblioteca GMP (*GNU Multiple Precision Arithmetic Library*) versão 4.2.4 [72];
- Biblioteca PBC (*Pairing Based Cryptography*) versão 0.4.18 [73];
- Biblioteca OpenSSL 0.9.8h;
- Biblioteca Libmilter 8.77;
- interface de rede ethernet 100Mbps.

Como cliente SMTP foi usado um computador com as seguintes características:

- Processador Pentium 4 3.2GHz;
- Cache L1 de 512KB;
- Memória RAM de 1GB;
- Sistema operacional Linux kernel 2.6.24-5-smp;
- Distribuição Slackware 12.1;
- MTA utilizado foi o sendmail 8.14.2.;
- interface de rede ethernet 100Mbps.

4.3 Aspectos de Implantação de Assinatura Digital Baseada em Identidade

Os processos de assinatura baseados em identidade, em geral, são divididos em quatro etapas: configurar, gerar, assinar e verificar.

Na etapa de configuração, a CA que é entidade confiável que irá gerar as chaves secretas para os usuários, deve produzir os parâmetros públicos que serão divulgados a todos que forem verificar a autenticidade das assinaturas. Esta divulgação deve garantir que o um agente malicioso não consiga divulgar parâmetros falsos. O CA também produzirá a chave secreta geral que será usada para calcular as chaves privadas.

Na etapa gerar, o usuário, para enviar uma mensagem, requisita ao CA uma chave secreta válida para um período determinado. Caso ele envie mensagens com diferentes endereços de remetentes, dever requisitar diferentes chaves. O CA recebe a identidade juntamente com a data de expiração para a qual ele deve calcular a chave secreta e retorna a chave ao usuário.

Na etapa assinar, o MUA deve montar uma cadeia de caracteres que represente a mensagem e assinar com a chave secreta recebida do CA. Esta cadeia de caracteres deve conter informações que a diferencie de outras mensagens, para evitar o ataque de repetição. O MUA então, assina a cadeia de caracteres utilizando um algoritmo IBS.

Na etapa verificar, o MTA, ao receber a mensagem, coleta as seguintes informações: endereço do remetente, composição da assinatura, domínio do remetente, assinatura parâmetros públicos. O MTA determina qual é a chave pública associada ao remetente e, juntamente com os parâmetros públicos, procede a verificação.

4.3.1 Configuração de Parâmetros Gerais

O problema de autenticação entre duas partes, sem a presença de uma terceira parte confiável, ainda não apresenta uma solução satisfatória [25]. A necessidade da presença do CA no esquema proposto neste trabalho justifica-se, pelo fato de que é possível calcular uma chave secreta a partir da chave pública desde que se conheça o valor secreto s . Conforme visto na Seção 2.3, um oponente pode escolher um valor de s' e divulgar P' e P'_{PUB} . O oponente pode então assinar uma mensagem que será validada se a verificação for feita com os parâmetros forjados. O destinatário necessita dos valores P e P_{PUB} divulgados pelo CA; caso contrário, o ataque do oponente funcionará.

Assim, pode-se montar um algoritmo para gerar os parâmetros gerais de um domínio.

-
1. Gerar de maneira aleatório um número $s \in \mathbb{Z}_q$ de N bits. O valor de N escolhe-se de acordo com nível de segurança. O valor de 160 bits mostra-se adequado atualmente [33, 43];
 2. Escolher aleatoriamente um ponto $P \in G$;
 3. Calcular $P_{PUB} = sP$.
-

Figura 4.7: Algoritmo de Geração de Parâmetros Gerais do Domínio

Para encontrar um ponto aleatório em G , escolhe-se uma coordenada x e tenta-se resolver a equação 2.8 e determinar a coordenada y . Caso não seja encontrada uma solução y , um novo ponto x é tentado até encontrar-se um ponto P . Para gerar os parâmetros s e P , utilizou-se a função `element_random()` da biblioteca `PBC_LIB` [73] e, para calcular o valor de P_{PUB} , utilizou-se a função `element_mul_zn()`

O trecho de código usado para gerar os parâmetros gerais é apresentado a seguir:

```

/* Geracao de parametros */
/* Escolhe Um valor aleatório para P */
element_random(P);
/* Escolhe Um valor aleatório para s */
element_random(s);
/* multiplica s por P e armazena em Ppub */
element_mul_zn(Ppub, P, s);

```

Para publicação dos parâmetros gerais do domínio, escolheu-se a estrutura do protocolo DNS. Um método interessante para essa publicação dos parâmetros é o uso de seletores [65]. Pode-se definir sub-domínios, como por exemplo `_safkeys.domain.com.br`, para que divulguem os parâmetros públicos P e P_{PUB} . A vantagem do uso de seletores é a flexibilidade na publicação dos parâmetros. Desta forma, evita-se a colisão com outros mecanismos que também utilizam o DNS como repositório público.

Entretanto, existe um limite de 256 caracteres para a informação que será armazenada nos servidores DNS [74]. Esta limitação de tamanho exige um método adequado para armazenar e transmitir os parâmetros públicos. A Figura 4.8 ilustra a impressão de tela da saída dos pontos aleatórios produzidos pela rotina de geração de parâmetros. Observa-se que a quantidade de caracteres necessários para publicar a informação ultrapassa o limite imposto pelo protocolo DNS.

```

P =
[740439100930348591165189819504696937828260034264135684317214060
4360841108987160267707313778328777880113587099813701886574766124
22042014705385093947354251,
4113971993427947999070692489773996628581074125418328255748733874
2772416526033121069537124599574669178221897160381949216857080052
18872256322942018868323025]
Ppub=
[125426719142113791358668660470150746014497743980428827280407179
9755409054966288756941948306123741440594353414446908844900843514
932183688988734078985311145,
7323237796931249108947059588721937093849552268684304787003051613
1568424904139572363803449079205029799314471885378414695392931521
46216483662427638112860693]

```

Figura 4.8: Parâmetros Públicos P e P_{PUB} em Formato Legível

O aumento do tamanho armazenado nos servidores DNS não é possível, posto que implicaria alterações no protocolo e aplicativos relacionados, e uma mudança na infra-estrutura foge

aos objetivos deste trabalho. Por essa razão, deve-se mudar a forma como as coordenadas dos pontos são armazenadas e transmitidas.

Lynn [33] propõem um mecanismo para reduzir a representação dos parâmetros, em que é feita a transmissão somente da coordenada x do ponto. A coordenada y pode ser calculada posteriormente, porém como todo x possui dois valores possíveis de y (um positivo e outro negativo), existiriam quatro pontos possíveis para parâmetros do sistema: P , $-P$, P_{PUB} e $-P_{PUB}$. Estes pontos permitiriam quatro permutações possíveis, portanto, seriam necessárias quatro verificações. Devido ao *overhead* computacional esta idéia foi descartada.

Como outra possível solução, analisou-se a função de compressão elemento para *bytes*. A função reduz para aproximadamente a metade do tamanho necessário para armazenar o ponto, porém este número deve ser convertido para uma representação em caracteres no formato *base64*.

A conversão *base64* eleva o tamanho de armazenamento para um valor um pouco abaixo de 70% do tamanho original da chave. A Tabela 4.1 exemplifica a relação do tamanho das chaves com o tamanho dos pontos em bytes, com os tamanhos dos formatos reduzidos e os tamanhos dos formatos reduzidos em codificação *base64*. Estes valores foram obtidos da média dos valores obtidos da geração de 100 conjuntos de parâmetros.

Tabela 4.1: Comparativo no Tamanho de Armazenamento

Tamanho da chave	Tamanho de P	P reduzido	base64
(bits)	(bytes)	(bytes)	(bytes)
160	128	65	90
224	190	97	134
256	256	129	175

Diante da dificuldade de se divulgar parâmetros cuja chave exceda 160 bits em um espaço de 256 bytes, os parâmetros P e P_{PUB} são armazenados em seletores distintos. O valor P é armazenado no seletor `p_safkey.<dominio do remetente>` e o valor P_{PUB} é armazenado no seletor `pub_safkey.<dominio do remetente>`.

Com a finalidade de reduzir o tamanho utilizado para representar os pontos, utilizou-se a função da biblioteca PBC, `element_to_bytes_compressed()`.

A representação textual dos parâmetros gerais a serem divulgados pelos seletores é organizada pelas seguintes marcas:

v=:	Esta marca recebe a versão do esquema utilizado. Ex. v=saf1
h=:	Algoritmo hash utilizado. Ex. h=sha-1,h=sha256,h=sha512
i=:	Esquema IBS utilizado. Ex. i=hess,i=paterson
s=:	Tamanho da chave. Ex. s=160,s=224,s=256
P=:	Ponto P no formato base64. Ex. P=FiG2nuMn7dxwgn5DdCA+SB15h7MfUWfDYH5CnR1IR22W1b47zRaSNcrZxRz2F4iGuYAThMxTDVSr
Pub=:	Ponto P_{PUB} no formato base64. Ex. PUB=F1kIIbxWHHvuP7cqnsqgCJhXiK0RWX XcEZg7Qc8KqLu32a0TnFbb1nOWNjMul96EHTlPP3BIza5OFumYHNJutQE=

```

...

1D IN NS      @
1D IN A       127.0.0.1

p_safkey      1D IN TXT      "v=saf1 h=sha1 k=160 P=SORaPPzKN
mgv9F3p+SkN+fWBFenNLdzvpasKcVEK1os7v3Wge9oDBmgst4gIAyD7FE1cvHebwyEBgH6YK6qjwE="
pub_safkey    1D IN TXT      "v=saf1 h=sha1 k=160 Pub=Y4h5Q4I
Po1Zc9T7s0sUj4yXo3wd0E6N5D9ZdwcUQvhUBhKT7uCMrEjMTd/M5ypmBfp/nOS80xi3CgiSfXZbbKQ=
="

...

```

Figura 4.9: Alterações no Arquivo localhost.zone

Para configurar o aplicativo *Bind*, na distribuição *Linux Slackware*, basta editar o arquivo `/var/named/caching-example/localhost.zone`. A Figura 4.9 ilustra um exemplo das linhas a serem incluídas no arquivo de configuração. Após editar o arquivo é necessário executar o comando `/etc/rc.d/rc.bind restart` para aplicar as modificações feitas.

4.3.2 Servidor de Chaves

O servidor de chaves é uma aplicação que recebe uma solicitação de cálculo de chave secreta. O servidor aguarda uma conexão TCP em uma determinada porta. O cliente abre uma conexão com o servidor e envia chave pública ID e o servidor responde com os parâmetros $v =$, $h =$, $s =$, P e a chave secreta D_{ID} , conforme ilustrado na Figura 4.10

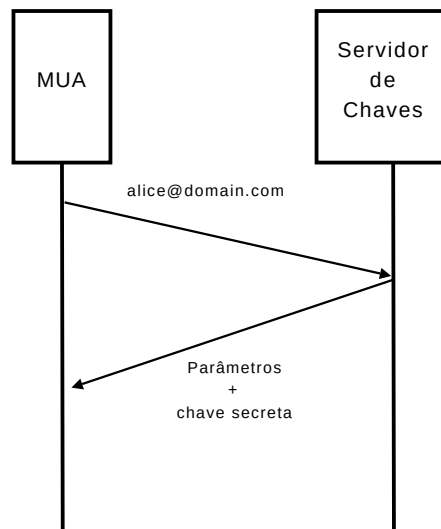


Figura 4.10: MUA Requisita Chave Secreta do Servidor de Chaves

O Algoritmo da Figura 4.11 exibe os procedimentos para gerar a chave secreta. Apesar de os valores P , h , i e k serem conhecidos publicamente, eles são informados pelo servidor para evitar que uma nova consulta do MUA ao repositório de parâmetros seja feita.

1. Recebe ID ;
2. Carrega valores gerais P , h , i e k ;
3. Carrega parâmetro secreto s ;
4. Calcula $Q_{ID} = H_2(ID)$;
5. Calcula $D_{ID} = sQ_{ID}$;
6. Retorna P , h , i , k e D_{ID} .

Figura 4.11: Algoritmo de Geração de Chaves Secretas

Os valores P e D_{ID} são representados em *bytes* codificados no formato base64, para facilitar a transmissão e o armazenamento.

4.3.3 Assinatura

Para assinar uma mensagem, o MUA deve antes solicitar uma chave secreta ao servidor de chaves, a qual será usada durante o resto do dia. Juntamente com a chave secreta, o cliente é informado do esquema IBS, tamanho da chave e algoritmo *hash*. A chave pública deve

ser criada a partir da concatenação das seqüências de caracteres dos campos especificados no cabeçalho `X-saf-key-fields` e o resultado da concatenação deve aparecer no cabeçalho `X-saf-key` que é a chave pública. A assinatura é montada de acordo com o Algoritmo da Figura 4.12.

1. Identificar os limites do corpo da mensagem e medir seu tamanho em *bytes*;
2. Calcular o *hash* do corpo da mensagem definido entre os limites;
3. Montar cadeia *C* a ser assinada;
4. Assinar a cadeia *C* usando a chave secreta e os parâmetros recebidos do servidor de chaves;
5. Incluir campos no cabeçalho da mensagem;

Figura 4.12: Algoritmo de Assinatura da Mensagem

Os campos a serem incluídos no cabeçalho e as respectivas descrições estão representados na Tabela 4.2.

Tabela 4.2: Campos do Cabeçalho da Mensagem Assinada

Campos	Conteúdo	Descrição
X-saf-fields:	From,To,Date,hash(body)	Campos que compõem a cadeia a ser assinada
X-saf-key-fields:	From;date	Campos que compõem a chave pública
X-saf-string:	alice@domain.com,bob@anotherdomain.com,2008-12-10, 5203cfa84f89a57c8dee370aea02ad27a8ab4c4a	Cadeia de caracteres a ser assinada
X-saf-key:	alice@dominio.com.br20081210	Chave pública
X-saf-blenght:	5000	Quantidade de <i>bytes</i> do corpo da mensagem
X-saf-sig-R:	dkjsfhkshfkjsahfoahfdssaskkhfsdaoilkajfklasdj =	Ponto <i>R</i> da assinatura
X-saf-sig-S:	ooihooiiooiuosafhdsgghdsydushbsudusdhisoihfbhsd=	Ponto <i>S</i> da assinatura

O valor de comprimento de corpo deve ser usado quando deseja-se que eventuais acréscimos ao corpo da mensagem sejam tolerados. Em caso de omissão considera-se que o corpo todo da mensagem será assinado.

Para desenvolver a assinatura da mensagem utiliza-se para o esquema de Paterson, o código a seguir.

```

...
/* inicializando elementos */
element_init_G1(P, pairing);
element_init_G1(Ppub, pairing);
element_init_Zr(s, pairing);
element_random(k);
element_mul_zn(R, P, k);
element_from_hash(t1, X-saf-sig, strlen(X-saf-sig));
element_mul_zn(t2, P, t1);
element_to_mpz(t3, R);
element_mul_mpz(t4, Did, t3);
element_add(t5, t4, t2);
element_invert(k, k);
element_mul_zn(S, t5, k);
element_printf("R = %B\n", R);
element_printf("S = %B\n", S);
...

```

Figura 4.13: Código para Assinatura IBS-Paterson

Para o desenvolvimento de uma rotina para o processo de assinatura de Hess, pode-se usar o código a seguir:

```

...
/* inicializando elementos */
element_init_G1(P, pairing);
element_init_G1(Ppub, pairing);
element_init_Zr(s, pairing);
element_random(P1);
element_random(k);
pairing_apply(t1, P1, P, pairing);
element_pow_zn(r, t1, k);
element_to_mpz(t2, r);
element_from_hash(t3, "Message123456789", 180000);
element_mul_mpz(v, t3, t2);
element_mul_zn(t4, Did, v);
element_mul_zn(t5, P1, k);
element_add(u, t4, t5);
printf("Signature of message \"Message\" is:\n");
element_printf("u = %B\n", u);
element_printf("v = %B\n", v);
...

```

Figura 4.14: Código para Assinatura IBS-Hess

4.3.4 Verificação de Assinatura

Atribuir função de verificar a assinatura da mensagem ao servidor, conforme mostrado, apresenta vantagens quando essa verificação é feita no destino, os usuários finais têm os recursos de conexão de último quilômetro desperdiçados, pois mesmo que disponha de mecanismos de verificação da autenticidade do endereço do remetente, os recursos de rede já foram usados. Por outro lado, com a verificação da autenticidade no servidor, este desperdício é evitado.

Para o sucesso da verificação no servidor, algumas funcionalidades devem ser contempladas: as chaves públicas dos remetentes devem ser fáceis de se obter; deve haver baixo impacto na carga de trabalho do servidor; não se deve alterar a infra-estrutura de e-mail existente.

O IBS mostra-se adequado para a tarefa, pois é possível ao servidor obter a chave pública do remetente da mensagem com dados contidos no cabeçalho da mensagem. Outros mecanismos como o RSA também podem enviar a chave pública associada à assinatura, porém para verificar se uma chave é autêntica exige-se certificação de chave, divulgação da chave da entidade certificadora e consulta a repositório de chaves revogadas.

Com o uso do IBS, a certificação é implícita, pois é fácil associar a chave pública ao seu dono, e a validade da assinatura pode estar incluída na própria chave, facilitando assim a definição de um prazo de expiração da chave.

Neste trabalho avaliaram-se dois esquemas de IBS: o primeiro proposto por Paterson [48] e o segundo por Hess [50]. A comparação da eficiência entre os dois esquemas, pode ser averiguada na Tabela 4.3.

Tabela 4.3: Comparativo entre os Esquemas IBS

	Hess	Paterson
Assinatura	$1E + 1M$	$1M + 1SM$
Verificação	$2E + 2P / 1E + 1P$	$2E + 2P / 2E + 1P$

E= exponenciação, M= multiplicação escalar, SM= multiplicação escalar simultânea, P= emparelhamento

Fonte:Hess [50]

Observa-se, pelo número de operações realizadas para assinar e verificar, que o esquema proposto por Hess é mais eficiente para a verificação, enquanto que o esquema proposto por Paterson é mais eficiente na assinatura da mensagem. O administrador do sistema, portanto, pode optar dependendo do enfoque desejado. Por este motivo, adotou-se como padrão, neste trabalho, o esquema proposto por Hess, devido ao menor impacto na carga do servidor.

O esquema proposto por Hess para a verificação da assinatura pode ser executado pelo código a seguir:

```

...
pairing_apply(t6, u, P, pairing);
element_neg(Ppub, Ppub);
pairing_apply(t7, Qid, Ppub, pairing);
element_pow_zn(t7, t7, v);
element_mul(r, t6, t7);
element_to_mpz(t2, r);
element_from_hash(t3, "Message123456789", 180000);
element_mul_mpz(t8, t3, t2);
if (!element_cmp(t8, v)) {
    printf("Signature is valid!\n");
} else {
    printf("Signature is invalid!\n");
}
...

```

Figura 4.15: Código para Verificação de Assinatura IBS-Hess

Já o esquema proposto por Paterson foi implementado com o código a seguir:

```

...
element_from_hash(t1, "Message123456123123123", 180000);
element_mul_zn(t7, P, t1);
pairing_apply(t6, P, t7, pairing);
pairing_apply(t8, Ppub, Qid, pairing);
element_to_mpz(t3, R);
element_pow_mpz(t9, t8, t3);
element_mul(t10, t6, t9);
pairing_apply(t11, R, S, pairing);
if (!element_cmp(t10, t11)) {
    printf("Signature is valid!\n");
} else {
    printf("Signature is invalid!\n");
}
...

```

Figura 4.16: Código para Verificação de Assinatura IBS-Paterson

Neste trabalho foi proposto o Algoritmo 4.17 para verificação da autenticidade do remetente, o qual foi aplicado o algoritmo no desenvolvimento do MILTER responsável pela verificação das assinaturas dos e-mails.

1. Extrair campos do cabeçalho da mensagem;
 2. Extrair domínio do remetente a partir do campo `From`;
 3. Consultar servidor DNS para o registro TXT de `p_safkey.<domínio do remetente>`;
 4. Consultar servidor DNS para o registro TXT de `pub_safkey.<domínio do remetente>`;
 5. Verificar a data de vencimento da chave pública;
 6. Calcular o hash do corpo da mensagem usando o algoritmo hash especificado no marcador “h” dos parâmetros do sistema;
 7. Montar a cadeia *C* a ser verificada de acordo com o campo do cabeçalho `X-saf-fields`;
 8. Calcular o hash da cadeia *C*;
 9. Montar a assinatura utilizando a chave pública, os parâmetros públicos e cadeia *C*;
 10. Comparar a assinatura criada com a divulgada nos campos `X-saf-sig-Re` e `X-saf-sig-S`;
 11. Aceitar ou rejeitar a assinatura.
-

Figura 4.17: Algoritmo de Verificação de Remetente

Com base neste algoritmo, construiu-se o código do MILTER (Apêndice A.2) que verifica a assinatura da mensagem. Para configurar o MILTER, foi necessário alterar o arquivo de configuração do *Sendmail*, para que o MTA pudesse comunicar-se com o MILTER.

A Figura 4.18 exibe as macros acrescentadas ao arquivo `sendmail.mc`. O arquivo `/var/run/saf_milter.sock` é usado para comunicação entre o *Sendmail* e o MILTER. O arquivo deve ser informado como parâmetro da aplicação `saf_milter`.

```
...  
INPUT_MAIL_FILTER('saf_milter', 'S=unix:/var/run/saf_milter.sock,  
T=S:4m;R:4m')  
define('confINPUT_MAIL_FILTERS', 'saf_milter')  
...
```

Figura 4.18: Código para Verificação de Assinatura IBS-Paterson

Para gerar o arquivo `sendmail.cf` a partir do arquivo `.mc` foi utilizado o seguinte comando:

```
m4 < arquivo.mc > sendmail.cf
```

O arquivo `sendmail.cf` localiza-se em `\etc\mail` e, para que as mudanças no arquivo de configuração tenham efeito, executa-se o comando:

```
\etc\rc.d\rc.sendmail restart
```

4.4 Testes e Resultados

Esta seção apresenta os resultados de alguns experimentos e aponta os principais aspectos de desempenho e segurança.

Os testes visam provar conceitos utilizados, verificar a eficiência do método proposto com relação a outros métodos consagrados, por meio da medida de tempos de verificação de assinatura. Os testes, também, visam verificar a capacidade da proposta de detectar remetentes forjados ou mensagens adulteradas, bem como verificar sua adequação ao uso com listas de discussão e servidores intermediários.

4.4.1 Testes de Desempenho

Para comparar a eficiência do processo de IBS escolhido para assinar e verificar uma mensagem, desenvolveu-se uma rotina em que realiza-se um número de 1000 assinaturas e verificações para os tamanhos de chaves de 160, 224 e 256 bits. Os valores P , s foram aleatórios

para cada rodada de assinatura e verificação. Os tempos médios obtidos estão nas Tabelas 4.4 e 4.5

Tabela 4.4: Tempos de Assinatura e Verificação Core 2 Duo 2GHz

Chave (bits)	Operação	Hess tempo em s	Paterson tempo em s
160	Assinatura	0,0317	0,0211
	Verificação	0,0186	0,0332
224	Assinatura	0,0747	0,0513
	Verificação	0,0489	0,0906
256	Assinatura	0,1463	0,0908
	Verificação	0,1028	0,1753

Observa-se que o processo proposto por Hess tem um tempo menor que o proposto por Paterson para verificação e maior para assinatura, conforme previsto. Razão por que recomenda-se o uso do IBS proposto por Hess para que o impacto no servidor SMTP seja reduzido.

Tabela 4.5: Tempos de Assinatura e Verificação Pentium 4 3.2GHz

Chave (bits)	Operação	Hess tempo em s	Paterson tempo em s
160	Assinatura	0,0447	0,0310
	Verificação	0,0247	0,0452
224	Assinatura	0,1088	0,0725
	Verificação	0,0678	0,1205
256	Assinatura	0,2109	0,1359
	Verificação	0,1422	0,2487

No trabalho de Adida et al. [20] mediu-se os tempos de assinatura e verificação exibidos na Tabela 4.6. Os tempos foram obtidos em uma máquina Intel Pentium 4 3.2GHz com 2GB de memória RAM com um sistema operacional Fedora Core Linux kernel 2.6.9. A Tabela 4.5 foi obtida em uma máquina Intel Pentium 4 3.2 GHz com 1GB de memória RAM rodando uma distribuição Linux Slackware 12.1. Estes ambientes permitem uma avaliação dos resultados deste trabalho.

Tabela 4.6: Tempos de Assinar e Verificar

Chave (bits)	Operação	RSA tempo em s
1024	Assinatura	0,037
	Verificação	0,037
2048	Assinatura	0,210
	Verificação	0,211

Fonte: Adida et al. [20]

Comparando-se com os valores obtidos por Adida et al. e os obtidos neste trabalho, nota-se que para a chave de 160 bits, cujo equivalente RSA é de 1024 bits, o tempo da operações é semelhante. Já para as chaves maiores, é clara a superioridade de desempenho dos processos IBS baseados em emparelhamentos e curvas elípticas.

Para avaliar o desempenho do processo de geração de chaves, desenvolveu-se uma rotina onde realiza-se um número de 1000 gerações de chaves secretas para os tamanhos de chaves de 160, 224 e 256 bits. Os resultados obtidos nas medidas, bem como os resultados de geração de chaves RSA de 1024 e 2048 bits encontram-se na Tabela 4.7.

Tabela 4.7: Tempos de Geração de Chaves Secretas

Chave IBS (bits)	Chave RSA (bits)	Operação	Hess tempo em s	Paterson tempo em s	RSA tempo em s
160	1024	Gerar	0,0222	0,0222	0,167
224	2048	Gerar	0,0223	0,0223	1,209
256		Gerar	0,0223	0,0223	

Fonte das medidas RSA: Adida et al. [20]

Observa-se que os tempos usados para geração da chave secreta praticamente não variou entre os processos Hess e Paterson, mesmo para diferentes tamanhos de chaves. Isto era esperado, uma vez que é necessário somente uma multiplicação de um inteiro por um ponto para gerar a chave secreta. Apesar de não causar impacto no servidor SMTP, o processo de gerar chaves tem clara vantagem quando comparado aos resultados obtidos no trabalho de Adida et al..

4.4.2 Testes de Segurança

Neste trabalho, realizaram-se testes com a finalidade de prova de conceito. Testaram-se mensagens válidas, mensagens assinadas porém com conteúdo alterado, mensagens válidas com diversas assinaturas. Foram submetidas ao MILTER, também mensagens inválidas, simulando um cenário de ataque. Dentre as mensagens inválidas, destacam-se as com assinaturas forjadas e repetidas.

Os cenários de ataque considerados são:

1. Mensagens com assinaturas forjadas por terceiros. Neste cenário enquadram-se assinaturas inventadas, assinaturas com chaves secretas de outros usuários e assinaturas de usuários cujas chaves secretas não foram criadas pelo CA.
2. Mensagens assinadas com alterações no conteúdo. Neste cenário, o oponente, de posse de uma mensagem assinada, tenta obter vantagem alterando o conteúdo da mensagem.

Os cenários de prova de conceito testados são:

1. Mensagem assinada. Um grupo de 50 mensagens assinadas por 5 remetentes diferentes foram enviadas para usuário do Servidor SMTP;
2. Mensagem assinada e repassada por outro MTA. Neste teste usou-se um MTA para o MUA diferente do MTA que verifica;
3. Mensagem repassada com alteração no subject. Após a assinatura da mensagem, alterou-se o Subject incluindo a cadeia {Spam};
4. Mensagem repassada com acréscimo de mensagem ao final do corpo.

Os testes mostraram que o esquema proposto identificou corretamente mensagens válidas e inválidas, adicionando um cabeçalho `X-Saf-Status:PASS/FAIL` nas mensagens. As tentativas de ataque foram detectadas pelo MILTER, tanto tentativas de forjar assinaturas como tentativas repetir assinaturas válidas com corpos alterados.

Testes de ataques com tentativas mais elaboradas como busca exaustiva, contra os elementos criptográficos, não foram planejados, pois os ataques teriam durações extensas ou custos demasiadamente elevados para os *spammers* ou fraudadores [75].

Capítulo 5

Conclusão

O alto volume de *spam* causa transtornos aos usuários. Pesquisadores estudam métodos para combater tais ameaças. Uma das frentes de estudo concentra-se em identificar o autor da mensagem e, por meio de um conjunto de listas, analisar sua reputação. Um dos desafios é evitar que as identidades dos remetentes sejam forjadas.

Tendo em vista esse desafio, foram desenvolvidos métodos com abordagens diversas, de acordo com as quais usuários finais verificam a identidade de outros usuários finais ou servidores verificam a identidade de outros servidores.

Um outro enfoque é proposto por este trabalho em que o servidor verifica a identidade dos usuários finais. Esta proposta apresenta algumas vantagens em relação às anteriores. Ao impedir que mensagens indesejadas cheguem até o usuário final, evita-se que:

- que recursos de rede sejam desperdiçados;
- recursos de armazenamento sejam consumidos pelas mensagens indesejadas, minimizando problemas com caixas postais de tamanho limitado. Como o recebimento de mensagens é assíncrono, eventualmente usuários podem ter mensagens legítimas recusadas por falta de espaço de armazenamento nas caixas postais;
- atacantes com acesso a uma rede específica forjem a identidade de usuários desta mesma rede.

Para viabilizar que um servidor faça a autenticação da identidade do remetente, o mecanismo de verificação de assinaturas deve:

- possuir um mecanismo simples para o servidor obter a chave pública do remetente;

- ter um custo computacional baixo para não impactar no desempenho do servidor;
- aproveitar a infra-estrutura de *e-mail* existente, sem que sejam necessário alterações nos aplicativos.

O mecanismo proposto por este trabalho, utilizou o IBS que permitiu ao servidor obter a chave pública do remetente com facilidade, sem a necessidade de uso de certificados, pois a certificação é implícita. Para prover um menor custo computacional, escolheram-se esquemas IBS baseados em emparelhamentos sobre curvas elípticas. E para possibilitar que a proposta funcionasse com a atual infra-estrutura de *e-mail*, optou-se pelo uso de MILTERs.

Como prova de conceito, desenvolveu-se um protótipo de um MILTER, bem como *scripts* e pequenas aplicações para gerar chaves, criar assinaturas e enviar mensagens para teste do MILTER.

Os testes de segurança mostraram que o protótipo foi capaz de identificar e verificar as assinaturas dos remetentes, e que resistiu a ataques simples de assinaturas forjadas e repetição de assinaturas válidas. Considerando-se que um sistema é seguro se o custo da quebra for superior ao da informação protegida [25], e que para evitar o *spam* basta que o custo do envio exceda os lucros recebidos [75], pode-se dizer que trata-se um sistema robusto.

Os testes de desempenho permitiram que se confirmassem as previsões teóricas relacionadas aos tempos dos processos propostos de assinatura e verificação. Também confirmaram um desempenho equivalente para chaves menores e um desempenho superior para chaves com maior nível de segurança. E, principalmente, os testes mostraram tempos praticáveis em ambientes de produção.

O aspecto negativo da proposta diz respeito ao tamanho de armazenamento dos parâmetros públicos que, mesmo com técnicas de compressão utilizadas, não apresentou resultados que permitissem sua divulgação dentro da limitação de 256 *bytes* do DNS. Assim, foi proposto um esquema em que são necessárias duas consultas aos registros DNS. Como as consultas dos parâmetros são feitas por domínios, e os servidores DNS armazenam os resultados de consultas anteriores, o impacto das consultas é pequeno.

Como proposta para trabalhos futuros, indica-se a implementação e testes comparativos de outros esquemas IBS, como o proposto por Cha e Cheon [76]. Sugere-se também o desenvolvimento de um mecanismo seguro e a construção de um servidor para distribuição de chaves secretas.

Entende-se ainda que seria interessante o desenvolvimento de uma proposta para equili-

brar a relação de irretratabilidade entre remetente e destinatário, como proposto por Borisov, Goldberg e Brewer [77], pois, claramente, o destinatário possui uma vantagem sobre o remetente.

Este trabalho apresentou também como contribuição um modelo onde utiliza-se o DNS para publicação de parâmetros públicos que funcionam como CA. Assim como o DNS, o CA pode ser organizado de maneira hierárquico e distribuída [17].

Finalmente, deve-se ressaltar que os objetivos da pesquisa de propor um mecanismo para autenticação do remetente, em que a verificação da identidade seja feita no âmbito do servidor, foi alcançado com o sucesso esperado. E em vista da importância do tema e dos resultados alcançados, torna o trabalho relevante.

REFERÊNCIAS

1. TANENBAUM, A. S. *Redes de Computadores*. 4a. ed. [S.l.]: Editora Campus, 1997.
2. SMALLBERG, D. *Survey of SMTP implementations*. IETF, sep 1983. RFC 876. (Request for Comments, 876). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc876.txt>>.
3. KLENSIN, J. *Simple Mail Transfer Protocol*. IETF, abr. 2001. RFC 2821 (Proposed Standard). (Request for Comments, 2821). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc2821.txt>>.
4. KLENSIN, J. *Simple Mail Transfer Protocol*. IETF, out. 2008. RFC 5321 (Draft Standard). (Request for Comments, 5321). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc5321.txt>>.
5. WATSON, B. Beyond Identity: Addressing Problems that Persist in an Electronic Mail System with Reliable Sender Identification. In: *CEAS*. <http://www.ceas.cc/papers-2004/140.pdf>: [s.n.], 2004. Acessado em 16/02/2009.
6. BELLOVIN, S. M. Spamming, Phishing, Authentication, and Privacy. *Commun. ACM*, ACM, New York, NY, USA, v. 47, n. 12, p. 144, 2004. ISSN 0001-0782.
7. TEMPLETON, B. *Origin of the Term "spam" to Mean Net Abuse*. Acessado em 16/02/2009. Disponível em: <<http://www.templetons.com/brad/spamterm.html>>.
8. MESSAGELABS. *Annual Email Security Report*. December 2007. Acessado em 10/06/2008. Disponível em: <<http://www.messagelabs.com>>.
9. MESSAGELABS. *Annual Email Security Report*. December 2008. Acessado em 16/02/2009. Disponível em: <<http://www.messagelabs.com>>.
10. _____. *Antiphishing.org*. Acessado em 20/11/2008. Disponível em: <<http://www.antiphishing.org>>.
11. JAKOBSSON, M. Modeling and Preventing Phishing Attacks. *Financial Cryptography '05*, 2005. Acessado em 16/02/2009. Disponível em: <http://www.informatics.indiana.edu/markus/papers/phishing_jakobsson.pdf>.

12. JAKOBSSON, M. The Human Factor in Phishing. *Privacy & Security of Consumer Information '07.*, 2007. Acessado em 16/02/2009. Disponível em: <<http://www.informatics.indiana.edu/markus/papers/aci.pdf>>.
13. HARRIS, E. *The Next Step in the Spam Control War: Graylisting*. 2003. Acessado em 16/02/2009. Disponível em: <<http://projects.puremagic.com/greylisting/whitepaper.html>>.
14. FIPS. *FIPS PUB 46-3: Data Encryption Standard (DES)*. [S.l.], out. 1999. Acessado em 16/02/2009. Disponível em: <<http://www.itl.nist.gov/fipspubs/fip186-2.pdf>>.
15. FIPS. *FIPS PUB 197: Advanced Encryption Standard (AES)*. [S.l.], nov. 2001. iv + 47 p. (FIPS PUB, v. 197). Acessado em 16/02/2009. Disponível em: <<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>>.
16. RSA LABS. *DES Challenge III*. Acessado 16/02/2009. Disponível em: <<http://www.rsa.com/rsalabs/node.asp?id=2108>>.
17. KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet: uma Abordagem Top-Down*. 3rd. ed. [S.l.]: Pearson Addison Wesley, 2006.
18. STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999. ISBN 0-13-869017-0.
19. CALLAS, J. et al. *OpenPGP Message Format*. IETF, nov. 2007. RFC 4880 (Proposed Standard). (Request for Comments, 4880). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc4880.txt>>.
20. ADIDA, B. et al. Lightweight Email Signatures. In: PRISCO, R. D.; YUNG, M. (Ed.). *SCN*. Springer, 2006. (Lecture Notes in Computer Science, v. 4116), p. 288–302. ISBN 3-540-38080-9. Acessado em 16/02/2009. Disponível em: <<http://dblp.uni-trier.de/db/conf/scn/scn2006.html/AdidaCHR06>>.
21. GUPTA, V. et al. Performance Analysis of Elliptic Curve Cryptography for SSL. In: *WiSE '02: Proceedings of the 1st ACM workshop on Wireless security*. New York, NY, USA: ACM, 2002. p. 87–94. ISBN 1-58113-585-8. Acessado em 16/02/2009.
22. RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. *A Method for Obtaining Digital Signatures and Public-key Cryptosystems*. [S.l.], 1978. v. 21, n. 2, 120-126 p.
23. FIPS. *FIPS PUB 186: Digital Signature Standard (DSS)*. [S.l.], dez. 2000. (FIPS PUB). Acessado em 16/02/2009. Disponível em: <<http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>>.
24. SHIREY, R. *Internet Security Glossary, Version 2*. IETF, ago. 2007. RFC 4949 (Informational). (Request for Comments, 4949). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc4949.txt>>.
25. STALLINGS, W. *Criptografia e Segurança de Redes*. 4ª edição. ed. [S.l.]: Pearson, 2007.
26. KOBLITZ, N. *A Course in Number Theory and Cryptography*. 2nd. ed. New York: Springer-Verlag, 1994. ISBN 0-387-94293-9.

27. SHOUP, V. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2008. Acessado em 16/02/2009. Disponível em: <<http://www.shoup.net/ntb/ntb-v2.pdf>>.
28. KOBLITZ, N. Elliptic Curve Cryptosystems. *Mathematics of Computation*, v. 48, p. 203–209, 1987. ISSN 0025–5718.
29. SCHNEIER, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*. [S.l.]: Wiley, 1995. Paperback. ISBN 0471117099.
30. HANKERSON, D.; MENEZES, A.; VANSTONE, S. *Guide to Elliptic Curve Cryptography*. 1st. ed. [S.l.]: Springer, 2004.
31. MILLER, V. S. Elliptic Curves and their Use in Cryptography. In: *DIMACS Workshop on Unusual Applications of Number Theory*. [S.l.: s.n.], 1997.
32. FERNANDES, A. Elliptic Curves Cryptography. *Dr. Dobb's Journal*, dezembro 1999. Acessado em 16/02/2009. Disponível em: <<http://www.ddj.com/184411133?pgno=1>>.
33. LYNN, B. *On the Implementation of Pairing-Based Cryptosystems*. Tese (Doutorado) — Stanford University, Palo Alto, CA, 2007. Acessado em 16/02/2009. Disponível em: <<http://crypto.stanford.edu/abc/thesis.pdf>>.
34. BONEH, D.; FRANKLIN, M. Identity-Based Encryption from the Weil Pairing. *Lecture Notes in Computer Science*, v. 2139, p. 213–??, 2001. Acessado em 16/02/2009. Disponível em: <<http://citeseer.ist.psu.edu/boneh01identitybased.html>>.
35. MENEZES, A.; VANSTONE, S.; OKAMOTO, T. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In: *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1991. p. 80–89. ISBN 0-89791-397-3.
36. FREY, G.; RÜCK, H.-G. A Remark Concerning M-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Math. Comput.*, American Mathematical Society, Boston, MA, USA, v. 62, n. 206, p. 865–874, 1994. ISSN 0025-5718.
37. DIFFIE, W.; HELLMAN, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976. Acessado em 16/02/2009. Disponível em: <<http://citeseer.ist.psu.edu/340126.html>>.
38. MAO, W. *Modern Cryptography: Theory and Practice*. [S.l.]: Prentice Hall Professional Technical Reference, 2003. ISBN 0130669431.
39. KOCHER, P.; E, J. J.; JUN, B. Differential Power Analysis. In: . [S.l.]: Springer-Verlag, 1999. p. 388–397.
40. POINTCHEVAL, D. *How to Encrypt Properly with RSA*. 2002. Acessado em 16/02/2009. Disponível em: <http://www.di.ens.fr/users/pointche/Documents/Papers/2002_cryptobytes.pdf>.
41. NIPPON TELEGRAPH AND TELEPHONE CORP; UNIVERSITY OF BONN; ECOLE POLYTECHNIQUE DE LAUSANNE. *New World Record for "Integer Factorization" Used to Verify Security of Public Key Cryptography*. 2007. Acessado em 16/02/2009. Disponível em: <<http://www.ntt.co.jp/news/news07e/0705/070521a.html>>.

42. JURISIC, A.; MENEZES, A. J. Elliptic Curves and Cryptography. *Dr. Dobbs's Journal*, 1997. Acessado 16/02/2009. Disponível em: <<http://www.ddj.com/184410167>>.
43. CERTICOM. *The Certicom ECC Challenge*. Acessado em 16/02/2009. Disponível em: <<http://www.certicom.com/index.php/the-certicom-ecc-challenge>>.
44. FIPS. *FIPS 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3*. [S.l.], October 2008. Acessado em 16/02/2009. Disponível em: <http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>.
45. WANG, X.; YIN, Y. L.; YU, H. Finding Collisions in the Full SHA-1. In: . [S.l.]: Springer, 2005. p. 17–36.
46. SHAMIR, A. Identity-Based Cryptosystems and Signature Schemes. In: *CRYPTO*. [s.n.], 1984. p. 47–53. Acessado em 16/02/2009. Disponível em: <<http://link.springer.de/link/service/series/0558/bibs/0196/01960047.htm>>.
47. COCKS, C. An Identity Based Encryption Scheme Based on Quadratic Residues. In: *Proceedings of the 8th IMA International Conference on Cryptography and Coding*. London, UK: Springer-Verlag, 2001. p. 360–363. ISBN 3-540-43026-1.
48. PATERSON, K. G. *ID-ased Signatures from Pairings on Elliptic Curves*. 2002. Cryptology ePrint Archive, Report 2002/004. Acessado em 16/02/2009. Disponível em: <<http://eprint.iacr.org/2002/004>>.
49. ELGAMAL, T. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31, n. 4, p. 469–472, July–1985.
50. HESS, F. Efficient Identity Based Signature Schemes Based on Pairings. In: *SAC '02: Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*. London, UK: Springer-Verlag, 2003. p. 310–324. ISBN 3-540-00622-2.
51. SEGALLER, S. *Nerds 2.0.1, A Brief History of the Internet*. [S.l.]: TV Books, 1998.
52. FREED, N.; BORENSTEIN, N. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. IETF, nov 1996. RFC 2045 (Draft Standard). (Request for Comments, 2045). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc2045.txt>>.
53. FREED, N.; BORENSTEIN, N. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. IETF, nov 1996. RFC 2046 (Draft Standard). (Request for Comments, 2046). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc2046.txt>>.
54. VENEMA, W. *Postfix Project*. Acessado em 16/02/2009. Disponível em: <<http://www.postfix.org/>>.
55. BERNSTEIN, D. *Qmail*. Acessado em 16/02/2009. Disponível em: <<http://www.qmail.org/>>.
56. HAZEL, P. *Qmail*. Acessado em 16/02/2009. Disponível em: <<http://www.exim.org/>>.
57. MICROSOFT CORPORATION. *Microsoft Exchange Server*. Acessado em 16/02/2009. Disponível em: <<http://www.microsoft.com/exchange/>>.

58. COSTALES, B.; ALLMAN, E. *Sendmail*. 2^a edition. ed. [S.l.]: O'Reilly & Associates, Inc, 1997.
59. SENDMAIL CONSORTIUM. *Sendmail*. Acessado em 16/02/2009. Disponível em: <<http://www.sendmail.org>>.
60. COSTALES, B. et al. *Sendmail*. 4^a edition. ed. [S.l.]: O'Reilly & Associates, Inc, 2007. ISBN 9780596510299.
61. VOLKERDING, P. *Slackware Linux*. Acessado em 16/02/2009. Disponível em: <<http://www.slackware.com>>.
62. KERNIGHAN, B. W.; RITCHIE, D. M. *The M4 Macro Processor*. Murray Hill, New Jersey, 1977.
63. Philip Zimmermann . *Pretty Goog Privacy*. Acessado em 16/02/2009. Disponível em: <<http://www.pgpi.org>>.
64. WONG, M.; SCHLITT, W. *Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1*. IETF, apr 2006. RFC 4408 (Experimental). (Request for Comments, 4408). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc4408.txt>>.
65. ALLMAN, E. et al. *DomainKeys Identified Mail (DKIM) Signatures*. IETF, may 2007. RFC 4871 (Proposed Standard). (Request for Comments, 4871). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc4871.txt>>.
66. FENTON, J. *Analysis of Threats Motivating DomainKeys Identified Mail (DKIM)*. IETF, sep 2006. RFC 4686 (Informational). (Request for Comments, 4686). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc4686.txt>>.
67. ASHIDANI, P. J. et al. Proposta de Framework para Autenticação de Remente. In: ABEAT. *Third International Conference of Forensic Computer Science*. Rio de Janeiro, Brazil: ABEAT, 2008. I, p. 116.
68. ATKINS, D.; AUSTEIN, R. *Threat Analysis of the Domain Name System (DNS)*. IETF, aug 2004. RFC 3833 (Informational). (Request for Comments, 3833). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc3833.txt>>.
69. ARENDS, R. et al. *DNS Security Introduction and Requirements*. IETF, mar 2005. RFC 4033 (Proposed Standard). (Request for Comments, 4033). Acessado em 16/02/2009. Disponível em: <<http://www.ietf.org/rfc/rfc4033.txt>>.
70. MCLAGAN, M.; JORDAN, M. J. *Linux Online*. Ogdensburg, New York: [s.n.]. Acessado em 16/02/2009. Disponível em: <<http://www.linux.org>>.
71. _____. *GNU Compiler Collection*. Acessado em 16/02/2009. Disponível em: <<http://gcc.gnu.org>>.
72. _____. *GNU Multiple Precision Arithmetic Library*. Acessado em 16/02/2009. Disponível em: <<http://gmplib.org/>>.
73. LYNN, B. *The Pairing-Based Cryptography Library*. Acessado em 16/02/2009. Disponível em: <<http://crypto.stanford.edu/pbc/>>.

74. LIU, C.; ALBITZ, P. *DNS e BIND*. [S.l.]: Editora Campus, 2001. ISBN 85-352-0869-0.
75. GOODMAN, J.; ROUNTHWAITE, R. Stopping Outgoing Spam. *ACM*, New York, 2004.
76. CHA, J. C.; CHEON, J. H. *An Identity-Based Signature from Gap Diffie-Hellman Groups*. 2002. Cryptology ePrint Archive, Report 2002/018. Acessado em 16/02/2009. Disponível em: <<http://eprint.iacr.org/>>.
77. BORISOV, N.; GOLDBERG, I.; BREWER, E. Off-the-Record Communication, or, Why Not to Use PGP. In: *WPES '04: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. New York, NY, USA: ACM, 2004. p. 77–84. ISBN 1-58113-968-3.

APÊNDICE A – Arquivos de Configurações, *Scripts* e Programas

A.1 Sendmail-milter.mc

```
dnl# This is the default sendmail .mc file for Slackware.
dnl# It is modified for using milter.
dnl#
include(`../m4/cf.m4')
VERSIONID(`default setup for Slackware Linux')dnl
OSTYPE(`linux')dnl
dnl# These settings help protect against people verifying email addresses
dnl# at your site in order to send you email that you probably don't want:
define(`confPRIVACY_FLAGS', `authwarnings,noverfy,noexpn,restrictqrun')dnl
dnl# Uncomment the line below to send outgoing mail through an external server:
dnl define(`SMART_HOST', `mailserver.example.com')
dnl# No timeout for ident:
define(`confTO_IDENT', `0')dnl
dnl# Enable the line below to use smrsh to restrict what sendmail can run:
dnl FEATURE(`smrsh', `/usr/sbin/smrsh')dnl
dnl# See the README in /usr/share/sendmail/cf for a ton of information on
dnl# how these options work:
FEATURE(`use_cw_file')dnl
FEATURE(`use_ct_file')dnl
FEATURE(`mailertable', `hash -o /etc/mail/mailertable.db')dnl
FEATURE(`virtusertable', `hash -o /etc/mail/virtusertable.db')dnl
FEATURE(`access_db', `hash -T<TMPF> /etc/mail/access')dnl
FEATURE(`blacklist_recipients')dnl
FEATURE(`local_procmail', `', `procmail -t -Y -a $h -d $u')dnl
FEATURE(`always_add_domain')dnl
FEATURE(`redirect')dnl
INPUT_MAIL_FILTER(`saf_milter', `S=unix:/var/run/saf_milter.sock, T=S:4m;R:4m')

define(`confINPUT_MAIL_FILTERS', `saf_milter')

dnl# Turn this feature on if you don't always have DNS, or enjoy junk mail:
dnl FEATURE(`accept_unresolvable_domains')dnl
EXPOSED_USER(`root')dnl
dnl# Also accept mail for localhost.localdomain:
LOCAL_DOMAIN(`localhost.localdomain')dnl
MAILER(local)dnl
MAILER(smtp)dnl
```

MAILER (procmail) dn1

A.2 Arquivo saf_milter.c

```
/*
saf_milter
filtro de e-mail utilizando framework de autenticação de
remetente.
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <syslog.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <openssl/sha.h>
#include "pbc/pbc.h"
#include "libmilter/mfapi.h"
#include "base64.h"

/* Forwards. */

static void usage(void);
static sfsistat saf_connect(SMFICTX* ctx, char* connhost, _SOCK_ADDR* connaddr);
static sfsistat saf_helo(SMFICTX* ctx, char* helohost);
static sfsistat saf_envfrom(SMFICTX* ctx, char** fromargs);
static sfsistat saf_envrcpt(SMFICTX* ctx, char** rcptargs);
static sfsistat saf_header(SMFICTX* ctx, char* name, char* value);
static sfsistat saf_eoh(SMFICTX* ctx);
static sfsistat saf_body(SMFICTX* ctx, unsigned char* bytes, size_t len);
static sfsistat saf_eom(SMFICTX* ctx);
static sfsistat saf_abort(SMFICTX* ctx);
static sfsistat saf_close(SMFICTX* ctx);
static void thread_data_destructor(void* data);

/* Globals. */

static char *fileParam;
static char *argv0;
static pthread_key_t key;
static int connections;
static char mail_from[100];
static char SignR[1024];
```

```
static char SignS[1024];
static char SignString[1024];
static char SignAssinatura[1024];
static char SignFields[1024];
static char mailfrom_domain[50];
static char bodyhash[512];
static int shaType;
static int versao;
static int algoritmo;

static struct smfiDesc smfilter = {
    "SAF_MILTER", // filter name
    SMFI_VERSION, // version code -- do not change
    SMFIF_ADDHDRS | SMFIF_CHGHDRS | SMFIF_CHGBODY | SMFIF_QUARANTINE, // flags
    saf_connect, // connection info filter
    saf_helo, // SMTP HELO command filter
    saf_envfrom, // envelope sender filter
    saf_envrcpt, // envelope recipient filter
    saf_header, // header filter
    saf_eoh, // end of header
    saf_body, // body block filter
    saf_eom, // end of message
    saf_abort, // message aborted
    saf_close // connection cleanup
};

int main(int argc, char** argv) {
    char* sockarg;
    char character;
    int count;
    FILE *fp;

    argv0 = strrchr(argv[0], '/');

    if (argv0 != (char*) 0)
        ++argv0;
    else
        argv0 = argv[0];

    connections = 0;

    if (argc != 3)
        usage();

    sockarg = argv[1];

    // ler o arquivo de parametros

    if ((fp = fopen(argv[2], "r")) == NULL) {
        (void) fprintf(stderr, "Erro ao obter arquivo de parametros: \"%s\"\n", argv[2]);
        exit(1);
    };

    // leitura do arquivo de parametros de pareamento
```

```

count = 0;
fileParam = malloc(1);
do {
    character = getc(fp);
    fileParam[count] = character;

    count++;
    fileParam = realloc(fileParam, count + 1);
} while (character != EOF);

fclose(fp);

if (pthread_key_create(&key, &thread_data_destructor)) {
    perror("pthread_key_create");
    exit(1);
}

openlog(argv0, 0, LOG_MAIL);

(void) smfi_setconn(sockarg);

// registrar
if (smfi_register(smfilter) == MI_FAILURE) {
    (void) fprintf(stderr, "%s: register failed\n", argv0);
    exit(1);
}

// iniciar o monitoramento do milter
if (smfi_main() == MI_FAILURE) {
    (void) fprintf(stderr, "%s: milter failed\n", argv0);
    exit(1);
}

// fechar limpar e sair
syslog(LOG_INFO, "exitting - %d connections", connections);
(void) pthread_key_delete(key);
closelog();
exit(0);

// limpar ponteiro do conteudo do arquivo
free(fileParam);
}

static void usage(void) {
    (void) fprintf(stderr, "usage: %s socket fileParam\n", argv0);
    exit(1);
}

/* Per-thread data structure. */
struct thread_data {
    int junk;

```

```

};

/* Per-connection data structure. */
struct connection_data {
    int connection_num;
    int messages;
    int recipients;
    int headers;
    int bodyblocks;
    size_t bodybytes;
    int aborts;
};

/* saf_connect - handle the initial TCP connection
**
** Called at the start of a connection. Any per-connection data should
** be initialized here.
**
** connhost: The hostname of the client, based on a reverse lookup.
** connaddr: The client's IP address, based on getpeername().
*/
static sfsistat saf_connect(SMFICTX* ctx, char* connhost, _SOCK_ADDR* connaddr) {
    struct thread_data* td;
    struct connection_data* cd;
    struct sockaddr_in* sin;

    td = (struct thread_data*) pthread_getspecific(key);

    if (td == (struct thread_data*) 0) {
        td = (struct thread_data*) malloc(sizeof (struct thread_data));

        if (td == (struct thread_data*) 0)
            return SMFIS_TEMPFAIL;
        if (pthread_setspecific(key, (void*) td)) {
            free((void*) td);
            return SMFIS_TEMPFAIL;
        }
    }

    cd = (struct connection_data*) malloc(sizeof (struct connection_data));
    if (cd == (struct connection_data*) 0)
        return SMFIS_TEMPFAIL;

    (void) smfi_setpriv(ctx, (void*) cd);
    cd->connection_num = ++connections;

    sin = (struct sockaddr_in*) connaddr;
    syslog(LOG_INFO, "con.##d - saf_connect( \"%s\" [%s] )",
        cd->connection_num, connhost, inet_ntoa(sin->sin_addr));

    cd->messages = 0;
    cd->aborts = 0;

    return SMFIS_CONTINUE;
}

```

```

}

/* saf_helo - handle the MAIL HELO
**
** Called at the start of a connection.
**
** helohost: The string passed to the HELO/EHLO command.
*/
static sfsistat saf_helo(SMFICTX* ctx, char* helohost) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_helo( \"%s\" )", cd->connection_num, helohost);

    return SMFIS_CONTINUE;
}

/* saf_envfrom - handle the MAIL FROM:<> command
**
** Called at the start of each message. There may be multiple messages
** in a connection. Any per-message data should be initialized here.
**
** The sender address is also known as the "envelope from" address.
**
** fromargs: Null-terminated SMTP command arguments; fromargs[0] is the
**           sender address.
*/
static sfsistat saf_envfrom(SMFICTX* ctx, char** fromargs) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_envfrom( \"%s\" )", cd->connection_num, fromargs[0]);
    strcpy(mail_from, fromargs[0]);
    strcpy(mailfrom_domain, extractDomain(fromargs[0]));

    ++cd->messages;

    cd->recipients = 0;
    cd->headers = 0;
    cd->bodyblocks = 0;
    cd->bodybytes = 0;

    return SMFIS_CONTINUE;
}

/* saf_envrcpt - handle a RCPT TO:<> command
**
** Called separately for each recipient of a message.
**
** rcptargs: Null-terminated SMTP command arguments; rcptargs[0] is the
**           recipient address.
*/
static sfsistat saf_envrcpt(SMFICTX* ctx, char** rcptargs) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_envrcpt( \"%s\" )", cd->connection_num,

```

```

rcptargs[0]);

++cd->recipients;

return SMFIS_CONTINUE;
}

/* saf_header - handle a header line
**
** Called separately for each header line in a message.
**
** name: Header field name.
** value: Header field value, including folded whitespace. The final CRLF
**        is removed.
*/
static sfsistat saf_header(SMFICTX* ctx, char* name, char* value) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_header( \"%s\", \"%s\" )",
           cd->connection_num, name, value);

    // guardar os valores das assinaturas do email
    if (strcmp("X-saf-sign-r", name) == 0) {
        strcpy(SignR, name);
    }
    else if (strcmp("X-saf-sign-s", name) == 0) {
        strcpy(SignS, name);
    }
    else if (strcmp("X-saf-string", name) == 0) {
        strcpy(SignString, name);
    }
    else if (strcmp("X-saf-key", name) == 0) {
        strcpy(SignAssinatura, name);
    }
    else if (strcmp("X-saf-field", name) == 0) {
        strcpy(SignFields, name);
    }

    ++cd->headers;

    return SMFIS_CONTINUE;
}

/* saf_eoh - handle the end of the headers
**
** Called once per message after all headers have been processed.
**
*/
static sfsistat saf_eoh(SMFICTX* ctx) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_eoh()", cd->connection_num);

    return SMFIS_CONTINUE;
}

```

```
/* saf_body - handle a block of body bytes
**
** The body of the message is filtered via zero or more calls to this
** routine. Most messages are handled with a single body call, but
** some do take more than one.
**
** bytes: Pointer to the block of body data.
** len: Length of the block.
*/
static sfsistat saf_body(SMFICTX* ctx, unsigned char* bytes, size_t len) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_body( %d )", cd->connection_num, (int) len);

    ++cd->bodyblocks;
    cd->bodybytes += len;

    switch (shaType) {
    1: SHA1(cd->bodyblocks, cd->bodybytes, bodyhash);
    break;
    160: SHA160(cd->bodyblocks, cd->bodybytes, bodyhash);
    break;
    512: SHA512(cd->bodyblocks, cd->bodybytes, bodyhash);
    break;
    }

    return SMFIS_CONTINUE;
}

static char* getDNSkey(char *key) {

    char consulta[255];
    char *resultado;
    char *chave;

    sprintf(consulta,
    "dig @%s -t txt %s.%s | grep %s.%s.",
    mailfrom_domain,
    key,
    mailfrom_domain,
    key,
    mailfrom_domain
    );

    resultado = system(consulta);

    shaType = extractParam("h");
    versao = extractParam("v");
    algoritmo = extractParam("i");
    chave = extractParam("key")

    return (chave);
}
```

```
static void saf_getP(element_t P) {

char p_key[255];

sprintf(p_key, getDNSkey("p_safkey"));
b64decode(pkey, pkey);

    element_from_bytes(P, p_key);
};

static void saf_getPpub(element_t Ppub) {

char p_key[255];

sprintf(p_key, getDNSkey("pub_safkey"));
b64decode(pkey, pkey);

    element_from_bytes(Ppub, p_key);
}

static void saf_getR(element_t R) {
    element_from_bytes(R, SignR);
}

// obtido do campo especifico do cabeçalho do e-mail

static void saf_getS(element_t S) {
    element_from_bytes(S, SignS);
}

/* ler apartir do campo correspondente do cabeçalho do e-mail quais
campos vao compor a string*/

static void saf_getFields(char *fields) {
    strcpy(fields, mailfrom);
}

static sfsistat saf_validademail(SMFICTX* ctx){
struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);
element_t elemento_assina;
element_t elemento_calcula;

if (DATE > (mail_date + 2)) {
return SMPFIS_REJECT;
}

char assinatura[1024];
sprintf(assinatura, "%s%s%s", mailfrom, DATE + 2, bodyhash);

element_from_hash(elemento_calcula, assinatura, strlen(assinatura));
    element_from_hash(elemento_assina, SignString, strlen(SignString));

    if (!element_cmp(elemento_assina, elemento_calcula)) {
```

```

        ret = SMFIS_REJECT;
    } else {
        ret = SMFIS_CONTINUE;
    }
}

static sfsistat saf_validatesign(SMFICTX* ctx) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    char fields[100];
    int ret;

    // variaveis utilizados para os calculos de paridade
    pairing_t pairing;
    element_t Ppub, P, R, S, Qid, t1, t6, t7, t8, t9, t10, t11;

    // variavel para guardar a ordem dos grupos
    // utilizados para o calculo
    mpz_t t3;
    mpz_init(t3);

    /* inicializar a variavel com o conteudo do arquivo de parametros obtido
    do servidor DNS */
    pairing_init_inp_buf(pairing, fileParam, strlen(fileParam));

    if (!pairing_is_symmetric(pairing)) {
        syslog(LOG_INFO, "con.##d - saf_validate(), only works with symmetric pairing",
            cd->connection_num);
        return SMFIS_TEMPFAIL;
    }

    // inicializar elementos que serão utilizados
    element_init_G1(P, pairing);
    element_init_G1(Ppub, pairing);
    element_init_G1(Qid, pairing);
    element_init_G1(R, pairing);
    element_init_G1(S, pairing);
    element_init_G1(t7, pairing);

    element_init_Zr(t1, pairing);

    element_init_GT(t6, pairing);
    element_init_GT(t8, pairing);
    element_init_GT(t9, pairing);
    element_init_GT(t10, pairing);
    element_init_GT(t11, pairing);

    // ler chaves do servidor DNS de chaves
    saf_getP(P);
    saf_getR(R);
    saf_getS(S);
    saf_getPpub(Ppub);
    saf_getID(Qid);

```

```

// validação da assinatura
saf_getFields(fields);
element_from_hash(t1, fields, strlen(fields));
element_mul_zn(t7, P, t1);
pairing_apply(t6, P, t7, pairing);
pairing_apply(t8, Ppub, Qid, pairing);

element_to_mpz(t3, R);
element_pow_mpz(t9, t8, t3);
element_mul(t10, t6, t9);

pairing_apply(t11, R, S, pairing);

if (!element_cmp(t10, t11)) {
    ret = SMFIS_REJECT;
} else {
    ret = SMFIS_CONTINUE;
}

//limpar elementos utilizados
element_clear(P);
element_clear(Ppub);
element_clear(Qid);
element_clear(R);
\chapter{Arquivo saf\_setup.c}
\label{apendice3}
\begin{scriptsize}
\input{apend4.tex}
\end{scriptsize}
element_clear(S);
element_clear(t1);
element_clear(t6);
element_clear(t7);
element_clear(t8);
element_clear(t9);
element_clear(t10);
element_clear(t11);
pairing_clear(pairing);

return ( ret);
}

/* saf_eom - handle the end of the message
**
** Called once per message after all body blocks have been processed.
** Any per-message data should be freed both here and in saf_abort().
**/
static sfsistat saf_eom(SMFICTX* ctx) {
    char status[10];
    int ret;
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    ret = saf_validatemail(ctx);
    switch (ret) {

```

```

        case SMFIS_CONTINUE: strcpy(status, "PASS");
            break;
        case SMFIS_REJECT: strcpy(status, "FAIL");
            break;
    }

    if (ret == SMFIS_CONTINUE) {
        // Efetuar as validações quando encontrar o final da mensagem de e-mail
        ret = saf_validatesign(ctx);
        switch (ret) {
            case SMFIS_CONTINUE: strcpy(status, "PASS");
                break;
            case SMFIS_REJECT: strcpy(status, "FAIL");
                break;
        }
    }

    if (smfi_addheader(ctx, "X-saf-Status", status) != MI_SUCCESS) {
        syslog(LOG_INFO, "con.##d - saf_eom(), Couldn't add header: X-saf-Status",
            cd->connection_num);
    }

    syslog(LOG_INFO,
        "con.##d - saf_eom(), %d recipients, %d headers, %d bodyblocks with %d bytes",
        cd->connection_num,
        cd->recipients,
        cd->headers,
        cd->bodyblocks,
        (int) cd->bodybytes
    );

    return ret;
}

/* saf_abort - handle the message being aborted
**
** May be called at any time during processing of a message, indicating
** that something went wrong. Any per-message data should be freed both
** here and in saf_eom().
**/
static sfsistat saf_abort(SMFICTX* ctx) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO,
        "con.##d - saf_abort(), %d recipients, %d headers, %d bodyblocks with %d bytes",
        cd->connection_num,
        cd->recipients,
        cd->headers,
        cd->bodyblocks,
        (int) cd->bodybytes
    );

    return SMFIS_CONTINUE;
}

```

```

/* saf_close - handle the connection being closed
**
** Called once at the end of a connection. Any per-connection data
** should be freed here.
*/
static sfsistat saf_close(SMFICTX* ctx) {
    struct connection_data* cd = (struct connection_data*) smfi_getpriv(ctx);

    syslog(LOG_INFO, "con.##d - saf_close(), %d messages", cd->connection_num, cd->messages);

    if (cd != (struct connection_data*) 0) {
        (void) smfi_setpriv(ctx, (void*) 0);
        free((void*) cd);
    }

    return SMFIS_CONTINUE;
}

static void thread_data_destructor(void* data) {
    struct thread_data* td = (struct thread_data*) data;

    free((void*) td);
}

```

A.3 Arquivo saf_setup.c

```

/*
Autor: Pedro Junior Ashidani
Rotina: Gerar chave secreta, gerar param publicos
Arquivo: setup.c
*/

#include <stdio.h>
#include <openssl/sha.h>
#include "pbc/pbc.h"
#include "pbc/pbc_time.h"
#include "base64.c"

#define arquivo_s    "arq_s.txt"
#define arquivo_P    "arq_P.txt"
#define arquivo_Ppub "arq_Ppub.txt"

int main(void) {

    FILE *fp;
    double time1, time2;
    pairing_t pairing;
    element_t s, Q, P, Ppub;

    // leitura dos parametros de emparelhamento

```

```
pairing_init_inp_str(pairing, stdin);
if (!pairing_is_symmetric(pairing)) {
    fprintf(stderr, "só funciona com emparelhamento simétrico\n");
    exit(1);
}

// pegar o tempo de início da geração
time1 = get_time();

// inicializando elementos
element_init_G1(P, pairing);
element_init_G1(Ppub, pairing);
element_init_G1(Q, pairing);
element_init_Zr(s, pairing);

// gerar elementos
element_random(P);
element_random(s);
element_mul_zn(Ppub, P, s);

printf("Elementos:\n");
element_printf("P      = %B\n", P);
element_printf("Ppub   = %B\n", Ppub);
element_printf("s       = %B\n", s);

// converter elementos para bytes
int length_P      = element_length_in_bytes_compressed(P);
int length_Ppub   = element_length_in_bytes_compressed(Ppub);
int length_s      = element_length_in_bytes(s);

char *b64_s;
char *b64_P;
char *b64_Ppub;

char *bt_s      = malloc(length_s);
char *bt_P      = malloc(length_P);
char *bt_Ppub   = malloc(length_Ppub);

element_to_bytes(bt_s, s);
element_to_bytes_compressed(bt_P, P);
element_to_bytes_compressed(bt_Ppub, Ppub);

// converter elementos para a base64 tornando-os transportáveis
b64_s      = b64encode(bt_s, length_s);
b64_P      = b64encode(bt_P, length_P);
b64_Ppub   = b64encode(bt_Ppub, length_Ppub);

printf("\nElementos convertidos para base64:\n");
printf("P      = %s\n", b64_P);
printf("Ppub   = %s\n", b64_Ppub);
printf("s      = %s\n", b64_s);

// gravação das chaves em arquivos
printf("\nGravando arquivos...\n");
```

```

// gravar arquivo "s"
if ((fp = fopen(arquivo_s, "w")) == NULL) {
    (void) fprintf(stderr, "Erro ao gravar arquivo de chaves s\n");
    exit(1);
}
fputs(b64_s, fp);
fclose(fp);
printf("arquivo \"%s\" gravado com sucesso.\n", arquivo_s);

// gravar arquivo "P"
if ((fp = fopen(arquivo_P, "w")) == NULL) {
    (void) fprintf(stderr, "Erro ao gravar arquivo de chaves p\n");
    exit(1);
}
fputs(b64_P, fp);
fclose(fp);
printf("arquivo \"%s\" gravado com sucesso.\n", arquivo_P);

// gravar arquivo de chave "Ppub"
if ((fp = fopen(arquivo_Ppub, "w")) == NULL) {
    (void) fprintf(stderr, "Erro ao gravar arquivo de chaves pub\n");
    exit(1);
}
fputs(b64_Ppub, fp);
fclose(fp);
printf("arquivo \"%s\" gravado com sucesso.\n", arquivo_Ppub);

// calcular o tempo de geração
time2 = get_time();
printf("\nTempo medio de geração = %fs\n", time2 - time1);

// limpar os elementos utilizados
element_clear(s);
element_clear(Q);
element_clear(P);
element_clear(Ppub);
pairing_clear(pairing);

printf("terminado.\n");

return 0;
}

```

A.4 Arquivo *assina.c*

```
]
```

```

#include <stdio.h>
#include <stdlib.h>

```

```
#include <pbcc.h>
#include <base64.h>
#include <openssl/sha.h>

char *main (int argc, char *argv[]) {

    char *chave;

    FILE *fp;
    char *file_param;
    char *file_sign;
    char *secret_key;
    char *saida1, saida2, result;
    char *data;
    char character;
    int count;

    element_t Ppub, s, P, R, k, S, Did, Qid, t1, t2, t4, t5, t6, t7, t8, t9, t10, t11;

    mpz_t t3;
    mpz_init(t3);

    if (argc != 5) {
        (void) fprintf(stderr, "usage: assina <mail> <file to sign> <file param>");
        exit(1);
    }

    strcpy(chave, argv[1]);

    // leitura do arquivo de parametros de pareamento
    if ((fp = fopen(argv[2], "r")) == NULL) {
        (void) fprintf(stderr, "Erro ao obter arquivo a ser assinado: \"%s\"\n", argv[2]);
        exit(1);
    };

    count = 0;
    file_sign = malloc(1);
    do {
        character = getc(fp);
        file_sign[count] = character;

        count++;
        file_sign = realloc(file_sign, count + 1);
    } while (character != EOF);

    fclose(fp);

    // leitura do arquivo de parametros
    if ((fp = fopen(argv[3], "r")) == NULL) {
        (void) fprintf(stderr, "Erro ao obter arquivo de parametros: \"%s\"\n", argv[2]);
        exit(1);
    };

    count = 0;
```

```
file_param = malloc(1);
do {
    character = getc(fp);
    file_param[count] = character;

    count++;element_from_hash(Qid, mail, strlen(mail));
    file_param = realloc(file_param, count + 1);
} while (character != EOF);

fclose(fp);

// leitura da chave secreta
if ((fp = fopen("s.param", "r")) == NULL) {
    (void) fprintf(stderr, "Erro ao obter arquivo de parametros: \"%s\"\n", argv[2]);
    exit(1);
};

count = 0;
secret_key = malloc(1);
do {
    character = getc(fp);
    secret_key[count] = character;

    count++;
    secret_key = realloc(secret_key, count + 1);
} while (character != EOF);

fclose(fp);

// hash do arquivo
SHA1(file_sign, strlen(file_sign), data);

// assinatura

pairing_init_inp_buff(pairing, file_param);
if (!pairing_is_symmetric(pairing)) {
    (void) fprintf(stderr, "only works with symmetric pairing\n");
    exit(1);
}

element_init_G1(P, pairing);
element_init_G1(Ppub, pairing);
element_init_G1(Qid, pairing);
element_init_G1(Did, pairing);
element_init_G1(R, pairing);
element_init_G1(S, pairing);
element_init_G1(t2, pairing);
element_init_G1(t4, pairing);
element_init_G1(t5, pairing);
element_init_G1(t7, pairing);

element_init_Zr(s, pairing);
element_init_Zr(k, pairing);
element_init_Zr(t1, pairing);
```

```

element_init_GT(t6, pairing);
element_init_GT(t8, pairing);
element_init_GT(t9, pairing);
element_init_GT(t10, pairing);
element_init_GT(t11, pairing);

element_from_hash(Qid, mail, strlen(mail));
element_mul_zn(Did, Qid, secret_key);

element_random(k);
element_mul_zn(R, P, k);

element_from_hash(t1, , strlen(file_sign));
element_mul_zn(t2, p, t1);

element_to_mpz(t3, r);
element_mul_mpz(t4, did, t3);

element_add(t5, t4, t2);
element_invert(k, k);

element_mul_zn(s, t5, k);

element_sprintf(saida1, "x-saf-sign-R: %b\n", R);
element_sprintf(saida2, "x-saf-sign-S: %b\n", S);

sprintf(result, "%s\n%s", saida1, saida2);

return ( result );
}

```

A.5 Arquivo envia-mail.sh

```

#!/bin/sh
# envio de email para fins de teste

# configuracoes
TELNET=/bin/telnet
SERVIDOR=localhost
PORTA=25
EMAIL_DE=root@localhost
EMAIL_PARA=root@localhost

#campos que irão compor a assinatura
CAMPOS="from"
KEY=""
KEYS_FIELDS=""
STRING=""

```

```
BLENGTH="5000"

# Assinaturas obtidas do pbc
ASSINATURA_R='pppppp'
ASSINATURA_S='pppppp'

(:
echo 'helo localhost'; sleep 1
echo 'mail from:' $EMAIL_DE; sleep 1
echo 'rcpt to:' $EMAIL_PARA; sleep 1
echo 'data'; sleep 1
echo 'subject:Envio de email para teste TCC'; sleep 1
echo 'X-saf-fields:' $CAMPOS; sleep 1
echo 'X-saf-key:' $KEY; sleep 1
echo 'X-saf-key-fields:' $KEYS_FIELDS; sleep 1
echo 'X-saf-string:' $STRING; sleep 1
echo 'X-saf-blength:' $BLENGTH; sleep 1
echo 'X-saf-sign-R:' $ASSINATURA_R; sleep 1
echo 'X-saf-sign-S:' $ASSINATURA_S; sleep 1
echo ''; sleep 1
cat $1; sleep 1
echo '.'; sleep 1
echo 'quit';
) | $TELNET $SERVIDOR $PORTA 1>> /dev/null 2>> /dev/null

echo ''
echo 'email enviado'
echo ''
```